



*Ninja  
Library  
Specification*





# *Ninja Library Specification*

## *Table of Contents*

---

1. System Functions .....	NLS-1
njAlphaMode .....	Sets Alpha Mode. .... NLS-2
njColorBlendingMode .....	Sets Color Blending Mode. .... NLS-3
njExitSystem .....	Performs system termination processing. .... NLS-5
njIgnoreTextureAlphaMode .....	Sets Texture Alpha Mode. .... NLS-6
njInitSystem .....	Initializes the system. .... NLS-7
njInitVertexBuffer .....	Allocates the buffers for registration of vertex data. .... NLS-11
njMipmapAdjust .....	Adjusts Mipmap Level of Textures. .... NLS-13
njModifierVolumeMode .....	Sets Modifier Volume Mode .... NLS-15
njPolygonCullingMode .....	Sets Polygon Culling Mode .... NLS-16
njPolygonCullingSize .....	Sets Polygon Size for Culling .... NLS-17
njSetBackColor .....	Sets background color. .... NLS-18
njSetVSyncFunction .....	Registers the vertical sync interrupt callback function. .... NLS-20
njSpecularMode .....	Sets Specular Mode .... NLS-22
njSuperSampleMode .....	Sets texture super sample. .... NLS-23
njTextureClampMode .....	Sets the texture clamp. .... NLS-24
njTextureFilterMode .....	Sets the texture filter. .... NLS-25
njTextureFlipMode .....	Sets the texture flip. .... NLS-27
njTextureShadingMode .....	Sets Texture Shading Mode .... NLS-28
njVersion .....	Gets the library version. .... NLS-29
njWaitVSync .....	Waits for a vertical interrupt. .... NLS-30

<b>2. Matrix Functions .....</b>	<b>NLS-33</b>
njAddMatrix .....	Performs matrix addition. .... NLS-35
njAddVector .....	Performs vector addition. .... NLS-36
njCalcPoint.....	Applies matrix conversion to an arbitrary point. .... NLS-37
njCalcVector.....	Applies matrix conversion to an arbitrary vector. .... NLS-39
njClearMatrix.....	Clears the matrix stack. .... NLS-41
njDetMatrix.....	Determines a matrix expression. .... NLS-42
njGetMatrix.....	Gets a copy of the current matrix. .... NLS-43
njInitMatrix.....	Initializes the matrix stack. .... NLS-44
njInnerProduct .....	Obtains the inner product of two vectors. .... NLS-46
njInvertMatrix .....	Obtains the inverse (reversed rows/columns) of a matrix. .... NLS-47
njMirror .....	Obtains the mirror image of an arbitrary boundary surface. .... NLS-48
njMultiMatrix.....	Performs matrix multiplication. .... NLS-50
njOuterProduct.....	Obtains the outer product of two vectors. .... NLS-51
njPopMatrix .....	Pops the matrix stack. .... NLS-52
njProject.....	Throws a parallel project onto an arbitrary picture plane. .... NLS-53
njProject2.....	Projects a transparent view onto an arbitrary picture plane. .... NLS-55
njProjectScreen.....	Projects an arbitrary point onto the screen. .... NLS-57
njPushMatrix .....	Pushes the matrix stack. .... NLS-59
njResMatrix.....	(Unsupported) .... NLS-60
njRotate.....	Rotates a matrix around an arbitrary axis. .... NLS-61
njRotateX.....	Applies a matrix that gives a rotation around the X axis. .... NLS-63
njRotateY.....	Applies a matrix that gives rotation around Y axis. .... NLS-64
njRotateZ.....	Applies a matrix that gives rotation around Z axis. .... NLS-65
njRotateXYZ.....	Applies a matrix that gives rotation around X, Y, and Z axes. .... NLS-66
njScale.....	Scales a matrix. .... NLS-68
njScaleV.....	Scales a matrix. .... NLS-70
njScalor .....	Returns the scalar of an arbitrary vector. .... NLS-71
njScalor2 .....	Returns the square of the scalar of an arbitrary vector. .... NLS-72
njSetMatrix.....	Copies an arbitrary matrix. .... NLS-73
njSubMatrix.....	Performs matrix subtraction. .... NLS-74
njSubVector .....	Performs vector subtraction. .... NLS-75
njTranslate .....	Applies a matrix that gives parallel translation along each axis. .... NLS-76
njTranslateV .....	Moves a matrix laterally. .... NLS-78
njTransposeMatrix .....	Transposes a matrix. .... NLS-80
njUnitMatrix .....	Converts an arbitrary matrix to a unit matrix. .... NLS-81
njUnitVector .....	Converts an arbitrary vector to a unit vector. .... NLS-82

<b>3. Collision Functions .....</b>	<b>NLS-83</b>
njCollisionCheckBB.....Checks Collision for 2 Hexahedron. ....	NLS-84
njCollisionCheckBC.....Checks collision for a hexahedron and a capsule. ....	NLS-86
njCollisionCheckBS.....Checking for collision for a hexahedron and a sphere. ....	NLS-88
njCollisionCheckCC.....Checks collision for two capsules. ....	NLS-90
njCollisionCheckSC.....Checks collision for a sphere and a capsule. ....	NLS-92
njCollisionCheckSS.....Checks collision for two spheres. ....	NLS-94
njDistanceL2L.....Returns the distance between two lines. ....	NLS-96
njDistanceL2PL.....Returns the distance between a line and a plane. ....	NLS-97
njDistanceP2L.....Returns the distance between a point and a line. ....	NLS-99
njDistanceP2P.....Returns the distance between two points. ....	NLS-101
njDistanceP2PL.....Returns the distance between a point and a line. ....	NLS-102
njDistancePL2PL.....Returns the distance between two planes. ....	NLS-104
njGetPnaneNormal.....Finds the vector that is normal to a plane. ....	NLS-105
njGetPnaneNormal2.....Finds the vector that is normal to a plane. ....	NLS-107
njIsParalellL2L.....Returns whether two lines are parallel. ....	NLS-109
njIsParalellL2PL.....Returns whether a line and a plane are parallel. ....	NLS-111
njIsParalellPL2PL.....Returns whether two planes are parallel. ....	NLS-113
 <b>4. Mathematical Functions .....</b>	 <b>NLS-115</b>
njAbs.....Returns an absolute value. ....	NLS-117
njArcCos.....Returns an arc cosine (ArcCos). ....	NLS-118
njArcCosec.....Returns an arc cosecant. ....	NLS-119
njArcCot.....Returns an arc cotangent. ....	NLS-120
njArcSec.....Returns an arc secant. ....	NLS-121
njArcSin.....Returns an arc sine. ....	NLS-122
njArcTan.....Returns an arc tangent. ....	NLS-123
njArcTan2.....Returns an arc tangent (ArcTan2). ....	NLS-124
njBezier..... <b>Bezier correction function</b> .....	NLS-125
njCardinalSpline..... <b>Cardinal spline correction function</b> .....	NLS-127
njCeil.....Returns the smallest integer not less than n (Ceiling Function). ....	NLS-129
njCombination..... <b>Combination function</b> .....	NLS-130
njCos.....Returns a cosine. ....	NLS-131
njCosec.....Returns the cosecant. ....	NLS-132
njCosech.....Returns the hyperbolic cosecant. ....	NLS-133
njCosh.....Returns the hyperbolic cosine. ....	NLS-134
njCot.....Returns the cotangent. ....	NLS-135
njCoth.....Returns the hyperbolic cotangent. ....	NLS-136
njCubicBezier..... <b>3D Bezier spline correction function</b> .....	NLS-137
njExp.....Returns exponents. ....	NLS-139
njFloor.....Returns the largest integer not greater than n (Floor Function). ....	NLS-140
njFraction.....Returns the decimal fraction. ....	NLS-141

njHermite.....	<b>Hermite spline correction function</b> .....	<b>NLS-142</b>
njHypot.....	Returns length of a hypotenuse. ....	NLS-144
njInvertSqrt.....	Returns the inverse square root. ....	NLS-145
njKochanekSpline .....	<b>Kochanek spline correction function</b> .....	<b>NLS-146</b>
njLinear.....	<b>Line shape correction function</b> .....	<b>NLS-148</b>
njLog.....	Returns the natural logarithm. ....	NLS-150
njLog10.....	Returns the base 10 logarithm. ....	NLS-151
njLog2.....	Returns the base 2 logarithm. ....	NLS-152
njOverhauserSpline .....	<b>Overhauser spline correction function</b> .....	<b>NLS-153</b>
njPow .....	Returns the power of a number. ....	NLS-155
njRandom .....	Generates a random number. ....	NLS-156
njRandomSeed .....	Sets the random number seed. ....	NLS-157
njRoundOff.....	Rounds down the decimal fraction. ....	NLS-158
njRoundUp .....	Rounds up the decimal fraction. ....	NLS-159
njSec .....	Returns a secant. ....	NLS-160
njSech.....	Returns a hyperbolic secant. ....	NLS-161
njSin .....	Returns a sine. ....	NLS-162
njSinh.....	Returns a hyperbolic sine. ....	NLS-163
njSqrt.....	Returns a square root. ....	NLS-164
njTan .....	Returns a tangent. ....	NLS-165
njTanh.....	Returns a hyperbolic tangent. ....	NLS-166

## 5. 2D Graphics Functions .....NLS-167

njDrawCircle2D .....	Draws circles on a 2D screen. ....	NLS-168
njDrawLine2D.....	Draws lines on a 2D screen. ....	NLS-171
njDrawPoint2D .....	Draws points on a 2D screen. ....	NLS-173
njDrawPolygon2D .....	Draws a polygon on a 2D screen. ....	NLS-175
njDrawTriangle2D .....	Draws triangles on a 2D screen. ....	NLS-177

## 6. 3D Graphics Functions .....NLS-179

njDrawLine3D.....	Draws lines in 3D space. ....	NLS-180
njDrawPoint3D .....	Draws points in 3D space. ....	NLS-182
njDrawPlygon3D .....	Draws a polygon in 3D space. ....	NLS-184
njDrawTriangle3D .....	Draws triangles in 3D space. ....	NLS-186

<b>7. Light Functions .....</b>	<b>NLS-189</b>
njSetLightAlpha .....	Sets the changes of alpha against material, whose light is set by njCreateLight. ....NLS-190
njCreateLight.....	Defines a light source type and registers a new light. ....NLS-192
njDeleteAllLight.....	This function deletes all the lights set at njCreateLight. ....NLS-196
njDeleteLight.....	Deletes a light created by njCreateLight. ....NLS-198
njLightAllOn .....	This function reflects the light set at njCreateLight (i.e. it turns on the lights). ....NLS-199
njLightAllOff.....	This function does not reflect all the lights set at njCreateLight (i.e. it turns off the lights). ....NLS-201
njLightOff .....	Deactivates a light created by njCreateLight (turns the light off). ....NLS-203
njLightOn.....	Activates a light created by njCreateLight (turns the light on). ....NLS-204
njMultiLightMatrix.....	Multiplies a matrix with a light matrix. ....NLS-206
njRotateLightX .....	Rotates a light matrix around the X axis. ....NLS-208
njRotateLightXYZ.....	Rotates a light matrix around the X, Y, and Z axes. ....NLS-210
njRotateLightY .....	Rotates a light matrix around the Y axis. ....NLS-212
njRotateLightZ .....	Rotates a light matrix around the Z axis. ....NLS-214
njSetLightAngle.....	Sets the limit angle of a light created with njCreateLight. ....NLS-216
njSetLightColor.....	Sets the color of light defined by njCreateLight. ....NLS-218
njSetLightDirection .....	Sets the direction of light defined by njCreateLight. ....NLS-220
njSetLightIntensity .....	Sets the intensity of light defined using njCreateLight. ....NLS-222
njSetLightLocation .....	Sets the location of light defined by njCreateLight. ....NLS-224
njSetLightRange .....	Sets the limit distance of a light created with njCreateLight. ....NLS-226
njSetUserLight .....	Assigns a user-defined light function to a light. ....NLS-228
njTranslateLight .....	Applies a matrix that gives parallel translation along each axis. ....NLS-233
njTranslateLightV .....	Moves a light matrix laterally according to a directional vector. ....NLS-235
njUnitLightMatrix .....	Unitizes a light matrix. ....NLS-237
 <b>8. Scroll Functions .....</b>	 <b>NLS-239</b>
njDrawScroll .....	Draws a 2D scroll surface. ....NLS-240

<b>9. Modeling Functions .....</b>	<b>NLS-243</b>
njSimpleDrawObject.....Draws objects. ....	NLS-243
njControl3D.....Controls the drawing surface for a 3D object. ....	NLS-244
njDrawModel.....Draws a model. ....	NLS-248
njDrawObject.....Draws an object. ....	NLS-250
njFastDrawModel.....Draws Models. ....	NLS-252
njFastDrawObject.....Draws objects. ....	NLS-254
njInit3D.....Initializing 3D system. ....	NLS-256
njSetConstantAttr.....Sets model attributes. ....	NLS-257
njSetConstantMaterial.....Sets model material data. ....	NLS-259
njSetDepthQueue.....Sets depth queue. ....	NLS-261
njSimpleDrawModel.....Draws models. ....	NLS-263
njSimpleDrawObject.....Draws objects. ....	NLS-264
 <b>10. View Functions .....</b>	 <b>NLS-267</b>
njCalcScreen.....Projects points in 3D space onto the screen, then finds the screen coordinates to which the points are projected. ....	NLS-269
njClip2D.....Specifies the drawing area on the screen. ....	NLS-271
njClipZ.....Specifies the limit values of near clipping and far clipping. ....	NLS-272
njForwardViewAbsolute.....Moves the view location in the direction of the view. (Absolute Move). ....	NLS-273
njForwardViewRelative.....Moves the view location along the viewline. (Relative Move). ....	NLS-275
njInitView.....Initializes the view. ....	NLS-277
njLookAtView.....Changes the view direction towards point (x, y, z). ....	NLS-278
njLookAtViewV.....Changes the view direction towards point (x, y, z). ....	NLS-280
njMultiViewMatrix.....Multiplies a view by a matrix. ....	NLS-282
njReturn2BaseView.....Returns the current view to the base view. ....	NLS-283
njRotateViewPosXAbsolute.....Rotates the view location around the X axis. (Absolute Rotation). ....	NLS-284
njRotateViewPosXRelative.....Rotates the view around the X axis (Relative Rotation). ....	NLS-286
njRotateViewPosYAbsolute.....Rotates the view location around the Y axis. (Absolute Rotation). ....	NLS-288
njRotateViewPosYRelative.....Rotates the view around the Y axis (Relative Rotation). ....	NLS-290
njRotateViewPosZAbsolute.....Rotates the view location around the Z axis. (Absolute Rotation). ....	NLS-292
njRotateViewPosZRelative.....Rotates the view around the Z axis (Relative Rotation). ....	NLS-294
njRotateViewX.....Rotates the view around the X axis (Absolute Rotation). ....	NLS-296
njRotateViewXAbsolute.....Rotates the view location around the X axis. (Absolute Rotation). ....	NLS-298
njRotateViewXRelative.....Rotates the view around the X axis (Relative Rotation). ....	NLS-300
njRotateViewXYZ.....Rotates the view around the X, Y, and Z axes (Absolute Rotation). ....	NLS-302
njRotateViewXYZAbsolute.....Rotates the line of view around the X, Y, and Z axes. (Absolute Rotation). ....	NLS-304



njRotateViewXYZRelative.....	Rotates the view around the X, Y, and Z axes (Relative Rotation).	.....NLS-306
njRotateViewY .....	Rotates the view around the Y axis (Absolute Rotation).	.....NLS-308
njRotateViewYAbsolute .....	Rotates the view line around the Y axis. (Absolute Rotation).	.....NLS-310
njRotateViewYRelative .....	Rotates the view around the Y axis (Relative Rotation).	.....NLS-312
njRotateViewZ .....	Rotates the view around the Z axis (Absolute Rotation).	.....NLS-314
njRotateViewZAbsolute .....	Rotates the view around the Z axis. (Absolute Rotation).	.....NLS-316
njRotateViewZRelative .....	Rotates the view around the Z axis (Relative Rotation).	.....NLS-318
njSetAspect.....	Sets the screen aspect ratio.	.....NLS-320
njSetBaseView .....	Sets the current view as the base view.	.....NLS-321
njSetPerspective.....	Sets the perspective in horizontal direction.	.....NLS-323
njSetScreen .....	Sets the screen.	.....NLS-324
njSetScreenDist.....	Sets the distance from the perspective to the screen.	.....NLS-325
njSetView .....	Specifies a user-defined view as the current view.	.....NLS-326
njTranslateView .....	Translates the view along the X, Y, and Z axes (Absolute Translation).	.....NLS-327
njTranslateViewAbsolute.....	Moves the view location along the X, Y, and Z axes. (Absolute Move).	.....NLS-329
njTranslateViewRelative .....	Translates the view along the X, Y, and Z axes (Relative Translation).	.....NLS-331
njTranslateViewV .....	Translates the view along the X, Y, and Z axes (Absolute Translation).	.....NLS-333
njTranslateViewVAbsolute.....	Moves the view location along the X, Y, and Z axes. (Absolute Move).	.....NLS-335
njTranslateViewVRelative .....	Translates the view along the X, Y, and Z axes (Relative Translation).	.....NLS-337
njUnitBaseViewVector .....	Converts the original view vector to a unit vector.	.....NLS-339
njUnitCurrentViewVector .....	Converts the view vector of the current view to a unit vector.	.....NLS-340
njUnitViewMatrix .....	Sets a unit matrix to the view matrix.	.....NLS-341
njUnitViewVector .....	Converts the view vector to a unit vector.	.....NLS-342

<b>11. Texture Functions</b>	<b>NLS-343</b>
njCalcTextureİl.Á8İl.Á8İl.Á8İl.Calculates the remaining texture memory size.	NLS-344
njGetTextureNumG.....Obtains the global index number of the current texture.	NLS-346
njInitTexture.....Sets the area used for storing texture information.	NLS-347
njLoadCacheTexture.....Sets the cache information area.	NLS-349
njLoadCacheTextureNum.....Loads a texture by texture number.	NLS-351
njLoadCacheTextureNumG.....Loads texture number globalIndex from cache memory into texture memory.	NLS-353
njLoadTexture.....Loads a texture.	NLS-355
njLoadTextureNum.....Loads textures.	NLS-358
njReleaseCacheTextureNum.....Release cache memory.	NLS-360
njReleaseCacheTextureAll.....Release all cache memory.	NLS-362
njReleaseCacheTextureNumG..Releases cache memory.	NLS-364
njReleaseTexture.....Releases texture memory.	NLS-366
njReleaseTextureNum.....Releases texture memory.	NLS-368
njReleaseTextureNumG.....Releases texture memory.	NLS-370
njReleaseTextureAll.....Release all texture memory.	NLS-372
njSetTexture.....Set current texture list.	NLS-374
njSetTextureInfo.....Set data into texture name structure.	NLS-376
njSetTextureName.....Set data into texture name structure.	NLS-379
njSetTextureNum.....Sets a texture number as the current texture.	NLS-381
njSetTextureNumG.....Sets the current texture to a global index number.	NLS-384
njReLoadTextureNum.....Reloads Textures.	NLS-386
njReLoadTextureNumG.....Reloads Textures.	NLS-388
njRenderTextureNum.....Renders Texture Area.	NLS-390
njRenderTextureNumG.....Renders at the Texture Area.	NLS-392
njSetRenderWidth.....Sets Stride Value.	NLS-393
 <b>12. Sprite Functions</b>	 <b>NLS-395</b>
njDrawSprite2D.....Draws 2D sprite.	NLS-396
njDrawSprite3D.....Draws 3D sprite.	NLS-400

<b>13. Debugging Functions .....</b>	<b>NLS-403</b>
njPrintB.....Displays a value in binary notation. ....	NLS-404
njPrintC .....	NLS-406
njPrintColor .....	NLS-407
njPrintD .....	NLS-408
njPrintF .....	NLS-410
njPrintH .....	NLS-412
njFrameBufferBmp.....Converts Frame Buffer To Bit Map Image. ....	NLS-414
njPrintSize .....	NLS-415
 <b>14. Special Effects Functions .....</b>	 <b>NLS-417</b>
njExcuteFade.....Excutes the fade effect. ....	NLS-418
njFadeDisable .....	NLS-420
njFadeEnable .....	NLS-422
njFogDisable .....	NLS-424
njFogEnable .....	NLS-426
njGenerateFogTable.....Creates Fog Table. ....	NLS-428
njGenerateFogTable2.....Creates Fog Table and Sets Density. ....	NLS-430
njGenerateFogTable3.....Creates Fog Table and Sets Density. ....	NLS-432
njSetFadeColor.....Specifies the fade color. ....	NLS-434
njSetFogColor.....Specifies the fog color. ....	NLS-436
njSetFogDensity .....	NLS-438
njSetFogTable .....	NLS-440

<b>15. Motion Functions</b>	<b>NLS-443</b>
njAction .....	Drawing motion. .... NLS-444
njActionLink.....	Links motions. .... NLS-446
njDrawMotion.....	Drawing Motion. .... NLS-448
njDrawMotionLink .....	Links motions. .... NLS-450
njDrawShapeMotion .....	Executes motion that includes shapes. .... NLS-452
njDrawShapeMotionLink.....	Links motions that include shapes. .... NLS-454
njFastAction .....	Draws Motion. .... NLS-456
njFastActionLink .....	Links motions. .... NLS-458
njFastDrawMotion .....	Draws Motion. .... NLS-460
njFastDrawMotionLink.....	Links motions. .... NLS-462
njFastDrawShapeMotion .....	Executes motion that includes shapes. .... NLS-464
njFastDrawShapeMotionLink .....	Links motions that include shapes. .... NLS-466
njGetMotionRotate.....	<b>This is a node motion information function (rotate).</b> .... NLS-468
njGetMotionScale.....	<b>This is a node motion info mation function (scale).</b> .... NLS-472
njGetMotionTranslate.....	<b>This is a node motion information function (move).</b> .... NLS-475
njMotionRotateXYZ .....	<b>This is a node motion function (rotate).</b> .... NLS-478
njMotionRotateZX.....	<b>This is a node motion function (ZXY rotate).</b> .... NLS-481
njMotionScale.....	<b>This is a node motion function (scale).</b> .... NLS-484
njMotionTransform.....	<b>This is a node motion function</b> .... NLS-487
njMotionTransformZXY.....	<b>This is a node motion function (ZXY rotate).</b> .... NLS-489
njMotionTranslate.....	<b>This is a node motion function (move).</b> .... NLS-492
njSetCurrentMotion .....	<b>Set parameters of the node motion function.</b> .... NLS-495
njSetNextMotionNode .....	<b>Update node of node motion function.</b> .... NLS-498
njSetMotionCallback.....	<b>Registering motion callback routine.</b> .... NLS-500
 <b>16. Memory Functions</b>	 <b>NLS-501</b>
njMemCopy .....	Memory Copy. .... NLS-502
njMemCopy4 .....	Memory Copy. (Word) .... NLS-503
njMemCopy2 .....	Memory Copy. (Long) .... NLS-504
 <b>17. Drawing Functions</b>	 <b>NLS-505</b>
njDrawPolygon .....	Draws polygons without textures. .... NLS-506
njDrawTexture .....	Draw texture polygons. .... NLS-508
 <b>18. Input Functions</b>	 <b>NLS-511</b>
njGetPeripheral.....	<b>Gets information of input device. (Peripheral).</b> .... NLS-512
njPrintPeripheralInfo.....	<b>Displays peripheral condition.</b> .... NLS-516



# 1. System Functions

---

## Contents

njAlphaMode .....	Sets Alpha Mode. ....	NLS-2
njColorBlendingMode .....	Sets Color Blending Mode. ....	NLS-3
njExitSystem .....	Performs system termination processing. ....	NLS-5
njIgnoreTextureAlphaMode .....	Sets Texture Alpha Mode. ....	NLS-6
njInitSystem .....	Initializes the system. ....	NLS-7
njInitVertexBuffer .....	Allocates the buffers for registration of vertex data. ....	NLS-11
njMipmapAdjust .....	Adjusts Mipmap Level of Textures. ....	NLS-13
njModifierVolumeMode .....	Sets Modifier Volume Mode .....	NLS-15
njPolygonCullingMode .....	Sets Polygon Culling Mode .....	NLS-16
njPolygonCullingSize .....	Sets Polygon Size for Culling .....	NLS-17
njSetBackColor .....	Sets background color. ....	NLS-18
njSetVSyncFunction .....	Registers the vertical sync interrupt callback function. ....	NLS-20
njSpecularMode .....	Sets Specular Mode .....	NLS-22
njSuperSampleMode .....	Sets texture super sample. ....	NLS-23
njTextureClampMode .....	Sets the texture clamp. ....	NLS-24
njTextureFilterMode .....	Sets the texture filter. ....	NLS-25
njTextureFlipMode .....	Sets the texture flip. ....	NLS-27
njTextureShadingMode .....	Sets Texture Shading Mode .....	NLS-28
njVersion .....	Gets the library version. ....	NLS-29
njWaitVSync .....	Waits for a vertical interrupt. ....	NLS-30

# njAlphaMode

Sets Alpha Mode.

---

## FORMAT

```
#include <Ninja.h>
void njAlphaMode( mode );
Int mode
```

## PARAMETERS

**mode**  
alpha mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets alpha mode. Following shows the modes that can be set:

ON	Sets alpha mode ON.
OFF	Sets alpha mode OFF.

## EXAMPLE

## NOTES

## RELATED TOPICS

# njColorBlendingMode

Sets Color Blending Mode.

## FORMAT

```
#include <Ninja.h>
void njColorBlendingMode( target, mode )
Int target
Int mode
```

## PARAMETERS

### target

Mode setting object

### mode

Blending mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets blending mode.
- Targets that can be set:

NJD_SOURCE_COLOR	source color
NJD_DESTINATION_COLOR	destination color

- Following shows the modes that can be set for each target:

NJD_COLOR_BLENDEING_BOTHINVALPHA	Regardless of the target specification, multiply source color by (1-As, 1-As, 1-As, 1-As) and destination color by (As, As, As, As) and blend.
NJD_COLOR_BLENDEING_BOTHSRCALPHA	Regardless of the target specification, multiply source color by (As, As, As, As) and destination color by (1-As, 1-As, 1-As, 1-As), and blend.
NJD_COLOR_BLENDEING_DESTALPHA	Multiply specified target by (Ad, Ad, Ad, Ad) and blend.
NJD_COLOR_BLENDEING_DESTCOLOR	Use the destination color for the specified target.
NJD_COLOR_BLENDEING_INVDESTALPHA	Multiply specified target by (1-Ad, 1-Ad, 1-Ad, 1-Ad) and blend.
NJD_COLOR_BLENDEING_INVDESTCOLOR	Use (1-Ad, 1-Rd, 1-Gd, 1-Bd) for specified target.
NJD_COLOR_BLENDEING_INVSRCALPHA	Multiply specified target by (1-As, 1-As, 1-As, 1-As) and blend.
NJD_COLOR_BLENDEING_INVSRCOLOR	Use (1-As, 1-Rs, 1-Gs, 1-Bs) for specified target.
NJD_COLOR_BLENDEING_SRCALPHA	Multiply specified target by (As, As, As, As) and blend.
NJD_COLOR_BLENDEING_SRCOLOR	Use source color for specified target.
NJD_COLOR_BLENDEING_ONE	Multiply specified target by (1, 1, 1, 1) and blend.
NJD_COLOR_BLENDEING_ZERO	Multiply specified target by (0, 0, 0, 0) and blend.

## EXAMPLE

## NOTES

s and d represent source color and destination color, respectively.

Example: As means alpha of source color.

## RELATED TOPICS



# njExitSystem

Performs system termination processing.

---

## FORMAT

```
#include <Ninja.h>
void njExitSystem( void )
```

## PARAMETERS

None

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Performs system termination processing.
- Always end by using this command.

## EXAMPLE

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
while(1) {
    .....
    .....
    if( ..... ) break;
    njWaitVSync();
}
njExitSystem();
```

## NOTES

## RELATED TOPICS

[njInitSystem\(\)](#)

# **njIgnoreTextureAlphaMode**

Sets Texture Alpha Mode.

---

## **FORMAT**

```
#include <Ninja.h>
void njIgnoreTextureAlphaMode( mode );
Int mode
```

## **PARAMETERS**

**mode**

alpha mode

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Sets texture alpha mode.
- The following modes can be set:

ON	Ignore the alpha bit of texture data
OFF	Do not ignore the alpha bit of texture data

## **EXAMPLE**

## **NOTES**

## **RELATED TOPICS**

# **njInitSystem**

Initializes the system.

---

## **FORMAT**

```
#include <Ninja.h>
void njInitSystem( mode, frame, count )
Int  mode
Int  frame
Int  count
```

## **PARAMETERS**

### **mode**

Screen mode (resolution)

Specifies the screen resolution.

### **frame**

Frame buffer mode.

### **count**

Number of frames

Specifies the number of frames in 1/60-sec units.

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Initializes the system and sets the specified screen resolution mode.
- Makes the 2D clip area the same as the screen size.
- Sets Z-clip to -2.0 to -60000.0.
- Sets the 3D screen's projection surface distance to 500.
- Sets both X and Y aspect to 1.0.
- Sets the color mode to NJD\_COLOR\_MODE\_NORMAL.
- Sets the number of frames in units of 1/60.
- For example, a setting 2 results in a frame change every 1/30 second.
- Frame changes are done with the njWaitVSync function.

Screen modes that can be set are as follows.

### Display Mode

NJD_RESOLUTION_VGA	VGA
NJD_RESOLUTION_320x240_NTSCNI	NTSC non interlace 60Hz
NJD_RESOLUTION_320x240_NTSCI	NTSC interlace 30Hz
NJD_RESOLUTION_640x240_NTSCNI	NTSC non interlace 60Hz
NJD_RESOLUTION_640x240_NTSCI	NTSC interlace 30Hz
NJD_RESOLUTION_640x480_NTSCI	NTSC interlace 30Hz
NJD_RESOLUTION_320x240_PALNI	PAL non interlace 50Hz
NJD_RESOLUTION_320x240_PALI	PAL interlace 25Hz
NJD_RESOLUTION_640x240_PALNI	PAL non interlace 50Hz
NJD_RESOLUTION_640x240_PALI	PAL interlace 25Hz
NJD_RESOLUTION_640x480_PALI	PAL interlace 25Hz

### Frame Buffer Mode

NJD_FRAMEBUFFER_MODE_RGB565
NJD_FRAMEBUFFER_MODE_RGB555
NJD_FRAMEBUFFER_MODE_ARGB4444
NJD_FRAMEBUFFER_MODE_ARGB1555
NJD_FRAMEBUFFER_MODE_RGB888
NJD_FRAMEBUFFER_MODE_ARGB8888

## EXAMPLE

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
```

Sets the screen resolution to 640x480, with a frame every 1/60 second.

## NOTES

Be sure to do this at the beginning of each program.

Depending on hardware configuration, some modes may not be available.

## RELATED TOPICS

`njExitSystem()`

`njWaitVSync()`

`njSetScreen()`

`njSetAspect()`

`njSetColorMode()`

`njClip2D()`

`njClipZ()`

## **njInitVertexBuffer**      Allocates the buffers for registration of vertex data.

---

### **FORMAT**

```
#include <Ninja.h>
void njInitVertexBuffer( op, om, tp, tm );
Uint32  op
Uint32  om
Uint32  tp
Uint32  tm
```

### **PARAMETERS**

**op:**

buffer size for non-transparent polygon registration

**om:**

buffer size for non-transparent modifier volume registration

**tp:**

buffer size for translucent polygon registration

**tm:**

buffer size for translucent modifier volume registration

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Allocates the buffers for registration of vertex data.
- Size for each is given # of bytes/4.

### **EXAMPLE**

```
njInitVertexBuffer( 100000, 0, 100000, 0 );
```

It reserves vertex buffers for non-transparent polygon and translucent polygon 400KB for each.

### **NOTES**

- When displaying debugg characters, reserve registration buffer for translucent polygons.

### **RELATED TOPICS**



## **njMipmapAdjust**

Adjusts Mipmap Level of Textures.

---

### **FORMAT**

```
#include <Ninja.h>
void  njMipmapAdjust( level )
Int  level
```

### **PARAMETERS**

**level**

Mipmap adjust level

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### FUNCTION

Sets 'D' parameter to adjust mipmap level.

Values can be set are 1 thru 15.

The following shows the actual D parameter values:

1	0.25
2	0.50
3	0.75
4	1.00
5	1.25
6	1.50
7	1.75
8	2.00
9	2.25
10	2.50
11	2.75
12	3.00
13	3.25
14	3.50
15	3.75

### EXAMPLE

### NOTES

### RELATED TOPICS

# **njModifierVolumeMode**

Sets Modifier Volume Mode

---

## **FORMAT**

```
#include <Ninja.h>
void njModifierVolumeMode( mode );
Int mode
```

## **PARAMETERS**

**mode**

modifier volume mode

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Sets modifier volume mode.
- The following modes can be set:

ON	Use modifier volume.
OFF	Do not use modifier volume.

## **EXAMPLE**

## **NOTES**

## **RELATED TOPICS**

# njPolygonCullingMode

Sets Polygon Culling Mode

---

## FORMAT

```
#include <Ninja.h>
void njPolygonCullingMode( mode );
Int mode
```

## PARAMETERS

**mode**

polygon culling mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets polygon culling mode
- The following modes can be set:

NJD_POLYGON_NOCULLING	Do not perform culling at all.
NJD_POLYGON_CULLINGSMALL	Perform culling for small polygons.
NJD_POLYGON_CULLINGACW	Perform culling to counter-clockwise polygons.
NJD_POLYGON_CULLINGCW	Perform culling to clockwise polygons.

## EXAMPLE

## NOTES

- In the NJD\_POLYGON\_CULLINGSMALL mode, the polygon size to be culled can be changed by the njPolygonCullingSize function.
- Default value: size = 0.01f.

## RELATED TOPICS

njPolygonCullingSize

# **njPolygonCullingSize**

Sets Polygon Size for Culling

---

## **FORMAT**

```
#include <Ninja.h>
void njPolygonCullingSize( size );
Float size
```

## **PARAMETERS**

**size**

Culling polygon size

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

Sets culling polygon size for NJD\_POLYGON\_CULLINGSMALL culling mode.

## **EXAMPLE**

## **NOTES**

## **RELATED TOPICS**

[njPolygonCullingMode](#)

# **njSetBackColor**

Sets background color.

## **FORMAT**

```
#include <Ninja.h>
void  njSetBackColor( color0, color1, color2 )
Uint32  color0
Uint32  color1
Uint32  color2
```

## **PARAMETERS**

**color0**

Upper left color

**color1**

Upper right color

**color2**

Bottom left color

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

Sets background color.

Colors should be specified with ARGB32bit.

Bottom right color is set automatically.

## EXAMPLE

This sample makes a background with blue in upper and black at the bottom side of the screen.

```
njSetBackColor( 0xff0000ff,0xff0000ff,0xff000000 );  
while(1) {  
    njDrawLine2D( ... );  
    ....  
    njWaitVSync();  
}
```

## NOTES

## RELATED TOPICS

[njWaitVSync\(\)](#)

## **njSetVSyncFunction**

Registers the vertical sync interrupt callback function.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetVSyncFunction( func )
void (*func)()
```

### **PARAMETERS**

**func**

Pointer to the vertical sync interrupt callback function

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Registers the vertical sync interrupt callback function.
- Only one such function can be registered.
- Specifying NULL deletes the registered function.



## EXAMPLE

```
static Int count=0;
void foo(void)
{
    count++;
}

njSetVSyncFunction( foo );

while(1) {
    ....
    njWaitVSync();
}
```

## NOTES

## RELATED TOPICS

[njWaitVSync\(\)](#)

# njSpecularMode

Sets Specular Mode

## FORMAT

```
#include <Ninja.h>
void njSpecularMode( mode );
Int mode
```

## PARAMETERS

**mode**

specular mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets specular mode.
- The following modes can be set:

ON	Set specular mode ON.
OFF	Set specular mode OFF.

## EXAMPLE

## NOTES

## RELATED TOPICS

# njSuperSampleMode

Sets texture super sample.

---

## FORMAT

```
#include <Ninja.h>
void  njSuperSampleMode( mode )
Int  mode
```

## PARAMETERS

**mode**

Texture super sample mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets texture super sample.
- The following modes can be set:

ON	Super sample filter ON
OFF	Super sample filter OFF

## EXAMPLE

## NOTES

## RELATED TOPICS

# njTextureClampMode

Sets the texture clamp.

## FORMAT

```
#include <Ninja.h>
void njTextureClampMode( mode )
Int mode
```

## PARAMETERS

**mode**

texture clamp mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets the texture clamp.
- The following modes can be set:

NJD_TEXTURECLAMP_NOCLAMP	No clamp
NJD_TEXTURECLAMP_CLAMP_V	V clamp
NJD_TEXTURECLAMP_CLAMP_U	U clamp
NJD_TEXTURECLAMP_CLAMP_UV	UV clamp

## EXAMPLE

## NOTES

## RELATED TOPICS

# njTextureFilterMode

Sets the texture filter.

## FORMAT

```
#include <Ninja.h>
void njTextureFilterMode( mode )
Int mode
```

## PARAMETERS

**mode**

texture filter mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets the texture filter.
- The following modes can be set:

NJD_TEXTUREFILTER_POINT_SAMPLE	Point sample
NJD_TEXTUREFILTER_BILINEAR	Bilinear filter
NJD_TEXTUREFILTER_TRILINEAR	Trilinear filter

## EXAMPLE

Sets the texture filter to bilinear filter.

```
njTextureFilterMode( NJD_TEXTUREFILTER_BILINEAR );
while(1) {
    njDrawLine2D( ... );
    ....
    njWaitVSync();
}
```

## NOTES

## RELATED TOPICS

`njWaitVSync()`

## njTextureFlipMode

Sets the texture flip.

---

### FORMAT

```
#include <Ninja.h>
void  njTextureFlipMode( mode )
Int   mode
```

### PARAMETERS

**mode**

texture flip mode

### RETURN VALUE

None

### ERROR VALUE

None

### FUNCTION

- Sets the texture flip.
- Modes can be set are

NJD_TEXTUREFLIP_NOFLIP	no flip
NJD_TEXTUREFLIP_FLIP_V	V flip
NJD_TEXTUREFLIP_FLIP_U	U flip
NJD_TEXTUREFLIP_FLIP_UV	UV flip

### EXAMPLE

### NOTES

### RELATED TOPICS

# njTextureShadingMode

Sets Texture Shading Mode

## FORMAT

```
#include <Ninja.h>
void njTextureShadingMode( mode );
Int mode
```

## PARAMETERS

**mode**

texture shading mode

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Sets texture shading mode.
- The following modes can be set:

NJD_TEX_SHADING_MODE_MODULATE	Multiply texture color by the shading effect color.
NJD_TEX_SHADING_MODE_DECALALPHA	Blend texture color with shading color.
NJD_TEX_SHADING_MODE_MODULATEALPHA	Multiply texture color by shading color.

## EXAMPLE

## NOTES

## RELATED TOPICS



## njVersion

Gets the library version.

---

### FORMAT

```
#include <Ninja.h>
Uint32 njVersion( void )
```

### PARAMETERS

None

### RETURN VALUE

None

### ERROR VALUE

None

### FUNCTION

- Gets the version number of the library.
- The version number is maintained as an 8-digit hexadecimal number.

8-digit hexadecimal number

```
00000000
| | | | | | | |
| | | | + + + + --- Build number
| | + + + + --- Revision
+ + + + + + + + --- Version
```

### EXAMPLE

```
version = njVersion();
```

### NOTES

### RELATED TOPICS

# **njWaitVSync**

Waits for a vertical interrupt.

---

## **FORMAT**

```
#include <Ninja.h>
void njWaitVSync()
```

## **PARAMETERS**

None

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Waits for a vertical interrupt.
- Performs a frame change and clears the screen.
- Calls the vertical sync interrupt callback function.

## **EXAMPLE**

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitVertexBuffer( 100000, 0, 100000, 0 );
while(1) {
    njDrawLine2D( ... );
    ....
    njWaitVSync();
}
```

## **NOTES**

## **RELATED TOPICS**

[njSetVSyncFunction\(\)](#)

[njSetBackColor\(\)](#)







## 2. Matrix Functions

---

### Contents

njAddMatrix.....	Performs matrix addition. ....	NLS-35
njAddVector.....	Performs vector addition. ....	NLS-36
njCalcPoint.....	Applies matrix conversion to an arbitrary point. ....	NLS-37
njCalcVector.....	Applies matrix conversion to an arbitrary vector. ....	NLS-39
njClearMatrix.....	Clears the matrix stack. ....	NLS-41
njDetMatrix.....	Determines a matrix expression. ....	NLS-42
njGetMatrix.....	Gets a copy of the current matrix. ....	NLS-43
njInitMatrix.....	Initializes the matrix stack. ....	NLS-44
njInnerProduct.....	Obtains the inner product of two vectors. ....	NLS-46
njInvertMatrix.....	Obtains the inverse (reversed rows/columns) of a matrix. ....	NLS-47
njMirror.....	Obtains the mirror image of an arbitrary boundary surface. ....	NLS-48
njMultiMatrix.....	Performs matrix multiplication. ....	NLS-50
njOuterProduct.....	Obtains the outer product of two vectors. ....	NLS-51
njPopMatrix.....	Pops the matrix stack. ....	NLS-52
njProject.....	Throws a parallel project onto an arbitrary picture plane. ....	NLS-53
njProject2.....	Projects a transparent view onto an arbitrary picture plane. ....	NLS-55
njProjectScreen.....	Projects an arbitrary point onto the screen. ....	NLS-57
njPushMatrix.....	Pushes the matrix stack. ....	NLS-59
njResMatrix.....	(Unsupported) ....	NLS-60
njRotate.....	Rotates a matrix around an arbitrary axis. ....	NLS-61
njRotateX.....	Applies a matrix that gives a rotation around the X axis. ....	NLS-63
njRotateY.....	Applies a matrix that gives rotation around Y axis. ....	NLS-64
njRotateZ.....	Applies a matrix that gives rotation around Z axis. ....	NLS-65
njRotateXYZ.....	Applies a matrix that gives rotation around X, Y, and Z axes. ....	NLS-66
njScale.....	Scales a matrix. ....	NLS-68
njScaleV.....	Scales a matrix. ....	NLS-70
njScalar.....	Returns the scalar of an arbitrary vector. ....	NLS-71

njScalar2 .....	Returns the square of the scalar of an arbitrary vector. ....	NLS-72
njSetMatrix.....	Copies an arbitrary matrix. ....	NLS-73
njSubMatrix .....	Performs matrix subtraction. ....	NLS-74
njSubVector .....	Performs vector subtraction. ....	NLS-75
njTranslate .....	Applies a matrix that gives parallel translation along each axis. ....	NLS-76
njTranslateV .....	Moves a matrix laterally. ....	NLS-78
njTransposeMatrix .....	Transposes a matrix. ....	NLS-80
njUnitMatrix .....	Converts an arbitrary matrix to a unit matrix. ....	NLS-81
njUnitVector .....	Converts an arbitrary vector to a unit vector. ....	NLS-82

## njAddMatrix

Performs matrix addition.

---

### FORMAT

```
#include <Ninja.h>

void njAddMatrix(*md, *ms)
NJS_MATRIX *md
NJS_MATRIX *ms
```

### PARAMETERS

**NJS\_MATRIX \*md**

destination matrix

**NJS\_MATRIX \*ms**

matrix to be added

### RETURN VALUE

None

### FUNCTION

- Performs matrix addition ( $[md] = [md] + [ms]$ ).
- If parameter md is NULL, the current matrix becomes the destination matrix.

### EXAMPLE

### NOTES

### RELATED TOPICS

NJS\_MATRIX

# **njAddVector**

Performs vector addition.

## **FORMAT**

```
#include <Ninja.h>

void njAddVector(*vd, *vs)
NJS_VECTOR *vd
NJS_VECTOR *vs
```

## **PARAMETERS**

**NJS\_VECTOR \*vd**

destination vector

**NJS\_VECTOR \*vs**

vector to be added

## **RETURN VALUE**

None

## **FUNCTION**

- Adds vectors and stores the result in vector vd.

**vd = vd + vs**

## **EXAMPLE**

## **NOTES**

## **RELATED TOPICS**

NJS\_VECTOR



## **njCalcPoint**

Applies matrix conversion to an arbitrary point.

---

### **FORMAT**

```
#include <Ninja.h>

void njCalcPoint(*m, *ps, *pd)
NJS_MATRIX *m
NJS_POINT3 *ps
NJS_POINT3 *pd
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_POINT3 \*ps**

arbitrary point

**NJS\_POINT3 \*pd**

point after conversion

### **RETURN VALUE**

None

### **FUNCTION**

- Applies matrix conversion to arbitrary point ps, then stores the post conversion coordinates of the point in pd.
- When parameter m is NULL, the current matrix is taken as the calculation matrix.

### EXAMPLE

The following rotates point ps(100, 0, 0) by 45 degrees around the Y axis.

```
NJS_MATRIX m;
NJS_POINT3 ps, pd;

ps.x = 100.f;
ps.y = 0.f;
ps.z = 0.f;
njUnitMatrix(&m);
njRotateY(&m, NJM_DEG_ANG(45));
njCalcPoint(&m, &ps, &pd);

% result %
pd(70.710701, 00.000000, -70.710701)
```

### NOTES

### RELATED TOPICS

## njCalcVector

Applies matrix conversion to an arbitrary vector.

---

### FORMAT

```
#include <Ninja.h>

void njCalcVector(*m, *vs, *vd)
NJS_MATRIX *m
NJS_VECTOR *vs
NJS_VECTOR *vd
```

### PARAMETERS

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_VECTOR \*vs**

arbitrary vector

**NJS\_VECTOR \*vd**

post-conversion vector

### RETURN VALUE

None

### FUNCTION

- Applies matrix conversion to arbitrary vector vs, then stores the vector resulting from conversion in vd.
- When parameter m is NULL, the current matrix is taken as the calculation matrix.

### EXAMPLE

The following rotates vector vs(100, 0, 0) 45 degrees around the Y axis.

```
NJS_MATRIX m;
NJS_VECTOR vs, vd;

vs.x = 100.f;
vs.y = 0.f;
vs.z = 0.f;
njUnitMatrix(&m);
njRotateY(&m, NJM_DEG_ANG(45));
njCalcVector(&m, &vs, &vd);

% result %
vd(70.710701, 00,000000, -70.710701)
```

### NOTES

### RELATED TOPICS

## **njClearMatrix**

Clears the matrix stack.

---

### **FORMAT**

```
#include <Ninja.h>

njClearMatrix()
```

### **PARAMETERS**

None

### **RETURN VALUE**

None

### **FUNCTION**

- Clears the matrix stack with the current view matrix.
- The current matrix pointer is restored to the base matrix pointer.
- See the View function View.txt file (in the Ninja root folder) for how to use this function.

### **EXAMPLE**

The following prepares and initializes a matrix stack and sets it with the current view.

```
NJS_MATRIX stack[10];

NJS_VIEW view;

njInitMatrix(stack, 10);

njInitView(&view);

njSetView(&view);

njClearMatrix();
```

### **NOTES**

### **RELATED TOPICS**

[njInitMatrix\(\)](#)  
[VIEW function](#)

# **njDetMatrix**

Determines a matrix expression.

---

## **FORMAT**

```
#include <Ninja.h>

float njDetMatrix(*m)
NJS_MATRIX *m
```

## **PARAMETERS**

**NJS\_MATRIX \*m**

calculation matrix

## **RETURN VALUE**

Determines the matrix

## **FUNCTION**

- Determines the matrix expression of arbitrary matrix m.
- When parameter m is NULL, the expression is determined from the current matrix.

## **EXAMPLE**

Determines the matrix expression of the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;
Float det;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
det = njDetMatrix(NULL);
njPopMatrix(1);
```

## **NOTES**

## **RELATED TOPICS**

## njGetMatrix

Gets a copy of the current matrix.

---

### FORMAT

```
#include <Ninja.h>

void njGetMatrix(*m)
NJS_MATRIX *m
```

### PARAMETERS

**NJS\_MATRIX \*m**

copy destination matrix

### RETURN VALUE

None

### FUNCTION

- Copies the current matrix to matrix m

### EXAMPLE

Copies the current matrix to matrix m.

```
NJS_MATRIX m;
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njGetMatrix(&m);
```

### NOTES

### RELATED TOPICS

## **njInitMatrix**

Initializes the matrix stack.

---

### **FORMAT**

```
#include <Ninja.h>

void njInitMatrix(*m, n flag)
NJS_MATRIX *m
Sint32 n
Int flag
```

### **PARAMETERS**

**m**

matrix stack

**n**

size of matrix stack

**Flag**

(true) use light matrix

(false) do not use light matrix

### **RETURN VALUE**

None

### **FUNCTION**

- Initializes the matrix stack.
- The matrix stack (an array) must be allocated by the user.
- Also, the stacked matrix is defined in the array of Float size 32.
- The 16(4x4) of the front half is used for the normal matrix operation and the 16(4x4) of the latter half is for the Light operation.
- In the case of using the 16 matrices of the front half for the Light operation, it is possible to skip the 16 matrix operations of the latter half by assigning 0 to the third argument.



## EXAMPLE

The following allocates a matrix stack of size 10 and initializes it.

```
NJS_MATRIX stack[10];  
njInitMatrix(stack, 10, 0);
```

## NOTES

- In this release, Model functions do not support light from the matrix.
- If you use Model functions, assign 0 to the flag.

## RELATED TOPICS

NJS\_MATRIX

## njInnerProduct

Obtains the inner product of two vectors.

---

### FORMAT

```
#include <Ninja.h>

Float njInnerProduct(*v1, *v2)
NJS_VECTOR *v1
NJS_VECTOR *v2
```

### PARAMETERS

**NJS\_VECTOR \*v1**

vector 1

**NJS\_VECTOR \*v2**

Vector 2

### RETURN VALUE

Value of inner product

### FUNCTION

- Obtains the inner product of two vectors.

**Return value=v1 \* v2**

### EXAMPLE

### NOTES

### RELATED TOPICS

NJS\_VECTOR

## **njInvertMatrix**

Obtains the inverse (reversed rows/columns) of a matrix.

---

### **FORMAT**

```
#include <Ninja.h>

void njInvertMatrix(*m)
NJS_MATRIX *m
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

source matrix

### **RETURN VALUE**

None

### **FUNCTION**

- Inverts the rows and columns of an arbitrary matrix.
- When parameter m is NULL, the current matrix is inverted.

### **EXAMPLE**

The following inverts the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_VECTOR v;

v.x = 0.5f;
v.y = 1.f;
v.z = 2.f;
njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njInvertMatrix(NULL);
njPopMatrix(1);
```

### **NOTES**

- If the arbitrary matrix is given as  $[M]$ , the result is  $[M] = [M]^{-1}$ .

### **RELATED TOPICS**

## **njMirror**      Obtains the mirror image of an arbitrary boundary surface.

---

### **FORMAT**

```
#include <Ninja.h>

void njMirror(*m, *pl)
NJS_MATRIX *m
NJS_PLANE *pl
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_PLANE \*pl**

boundary surface

### **RETURN VALUE**

None

### **FUNCTION**

- Obtains the mirror image of arbitrary boundary surface pl.
- When parameter m is NULL, the calculation is performed on the current matrix.

## EXAMPLE

The following obtains the mirror image through the origin of the boundary surface containing vector (0, 1, 1).

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_PLANE pl;

pl.px = 0.f;
pl.py = 0.f;
pl.pz = 0.f;
pl.vx = 0.f;
pl.vy = 1.f;
pl.vz = 1.f;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njMirror(NULL, &pl);
njPopMatrix(1);
```

## NOTES

## RELATED TOPICS

# njMultiMatrix

Performs matrix multiplication.

---

## FORMAT

```
#include <Ninja.h>

void njMultiMatrix(*md, *ms)
NJS_MATRIX *md
NJS_MATRIX *ms
```

## PARAMETERS

**NJS\_MATRIX \*md**

destination matrix

**NJS\_MATRIX \*ms**

calculation matrix

## RETURN VALUE

None

## FUNCTION

- Performs matrix multiplication. (  $[md] = [ms] * [md]$  )
- When parameter m is NULL, the current matrix is taken as the destination matrix.

## EXAMPLE

## NOTES

## RELATED TOPICS

NJS\_MATRIX

## njOuterProduct

Obtains the outer product of two vectors.

---

### FORMAT

```
#include <Ninja.h>

Float njOuterProduct(*v1, *v2, *ov)
NJS_VECTOR *v1
NJS_VECTOR *v2
NJS_VEDTOR *ov
```

### PARAMETERS

**NJS\_VECTOR \*v1**

vector 1

**NJS\_VECTOR \*v2**

vector 2

**NJS\_VEDTOR \*ov**

output vector

### RETURN VALUE

Absolute value of outer product

### FUNCTION

- Obtains the outer product of two vectors and returns the absolute value of the result.

**Return value** =  $|v1 * v2|$

### EXAMPLE

### NOTES

### RELATED TOPICS

NJS\_VECTOR

# njPopMatrix

Pops the matrix stack.

---

## FORMAT

```
#include <Ninja.h>

Bool njPopMatrix(n)
Uint32 n
```

## PARAMETERS

**Uint32 n**

Number of items to pop from the matrix stack

## RETURN VALUE

**TRUE**

Successful

**FALSE**

Failure (stack underflow)

## FUNCTION

- Pops n items from the matrix stack.

## EXAMPLE

The following pops two items from the matrix stack.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njPushMatrix(NULL);
njPopMatrix(2);
```

## NOTES

- The number of items popped is the same as the number of items pushed.

## RELATED TOPICS



## **njProject**      Throws a parallel project onto an arbitrary picture plane.

---

### **FORMAT**

```
#include <Ninja.h>

void njProject(*m, *pl)
NJS_MATRIX *m
NJS_PLANE *pl
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_PLANE \*pl**

picture plane

### **RETURN VALUE**

None

### **FUNCTION**

- Throws a parallel projection onto arbitrary picture plane pl.
- When parameter m is NULL, calculation is performed on the current matrix.

### EXAMPLE

The following throws a parallel projection through the origin onto the plane.

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_PLANE pl;

pl.px = 0.f;
pl.py = 0.f;
pl.pz = 0.f;
pl.vx = 0.f;
pl.vy = 1.f;
pl.vz = 1.f;
njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njProject(NULL, &pl);
njPopMatrix(1);
```

### NOTES

### RELATED TOPICS

## njProject2

Projects a transparent view onto an arbitrary picture plane.

---

### FORMAT

```
#include <Ninja.h>

void njProject2(*m, *pl, *v, *p)
NJS_MATRIX *m
NJS_PLANE *pl
NJS_POINT3 *v
NJS_POINT3 *p
```

### PARAMETERS

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_PLANE \*pl**

picture plane

**NJS\_POINT3 \*v**

projection viewpoint

**NJS\_POINT3 \*p**

center point of object to be projected

### RETURN VALUE

None

### FUNCTION

- Projects a transparent image onto arbitrary plane pl from viewpoint p.
- When parameter m is NULL, the current matrix is used for calculation.

### EXAMPLE

This projects a transparent image through the origin onto the plane containing vector (0, 1, 1) from point  $v(0, 100, 100)$ .

Center point of the object is at  $p(0, 10, 10)$ .

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_PLANE pl;
NJS_POINT v, p;

pl.px = 0.f;
pl.py = 0.f;
pl.pz = 0.f;
pl.vx = 0.f;
pl.vy = 1.f;
pl.vz = 1.f;
v.x = 0.f;
v.y = 100.f;
v.z = 100.f;
p.x = 0.f;
p.y = 10.f;
p.z = 10.f;
njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njProject2(NULL, &pl, &v, &p);
njPopMatrix(1);
```

### NOTES

### RELATED TOPICS

## **njProjectScreen**

Projects an arbitrary point onto the screen.

---

### **FORMAT**

```
#include <Ninja.h>

void njProjectScreen(*m, *p3, *p2)
NJS_MATRIX *m
NJS_POINT3 *p3
NJS_POINT2 *p2
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_POINT3 \*p3**

coordinates of arbitrary point

**NJS\_POINT2 \*p2**

point coordinates after projection

### **RETURN VALUE**

None

### **FUNCTION**

- Projects an arbitrary point onto the screen.
- When parameter m is NULL, the current matrix is used for calculation.

### EXAMPLE

The following projects point p3(100, 200, 300) onto the screen.

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_POINT3 p3;
NJS_POINT2 p2;

p3.x = 100.f;
p3.y = 200.f;
p3.z = 300.f;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njProjectScreen(NULL, &p3, &p2);
njPopMatrix(1);

% Result %
p2(153.333328, 573.333374)
```

### NOTES

### RELATED TOPICS

# **njPushMatrix**

Pushes the matrix stack.

---

## **FORMAT**

```
#include <Ninja.h>

Bool njPushMatrix(*m)
NJS_MATRIX *m
```

## **PARAMETERS**

**NJS\_MATRIX \*m**

matrix to be pushed

## **RETURN VALUE**

**TRUE**

successful

**FALSE**

failed (stack overflow)

## **FUNCTION**

- Pushes matrix m onto the matrix stack.
- When parameter m is NULL, this function pushes the current matrix.

## **EXAMPLE**

The following pushes a unit matrix onto the matrix stack.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(_unit_matrix_);
```

## **NOTES**

## **RELATED TOPICS**

# **njResMatrix**

(Unsupported)

---

**FORMAT**

**PARAMETERS**

**RETURN VALUE**

**FUNCTION**

**EXAMPLE**

**NOTES**

**RELATED TOPICS**



## **njRotate**

Rotates a matrix around an arbitrary axis.

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateX(*m, *v, ang)
NJS_MATRIX *m
NJS_VECTOR *v
Angle ang
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

calculation matrix

**NJS\_VECTOR \*v**

data for arbitrary axis

**Angle ang**

rotation angle

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates matrix m around an arbitrary axis.
- When parameter m is NULL, the current matrix is used for calculation.

### EXAMPLE

The following rotates the current matrix 90 degrees around the arbitrary axis containing vector  $v(1,1,1)$ .

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_VECTOR v;

v.x = 1.f;
v.y = 1.f;
v.z = 1.f;
njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njRotate(NULL, &v, NJM_DEG_ANG(90));
njPopMatrix(1);
```

### NOTES

- For a matrix  $[M]$  and multiplicative rotational matrix  $[R]$ ,  $[M] = [R] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

### RELATED TOPICS

## **njRotateX**

Applies a matrix that gives a rotation around the X axis.

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateX(*m, ang)
NJS_MATRIX *m
Angle ang
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

destination matrix

**Angle ang**

rotation angle

### **RETURN VALUE**

None

### **FUNCTION**

- Applies a matrix that gives rotation around the X axis to the matrix m.
- When parameter m is NULL, the current matrix will be the destination.

### **EXAMPLE**

The following applies a matrix that gives rotation of 90 degrees around the X axis to the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njRotateX(NULL, NJM_DEG_ANG(90));
njPopMatrix(1);
```

### **NOTES**

- For a matrix [M] and multiplicative rotational matrix [R],  $[M] = [R] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

### **RELATED TOPICS**

## **njRotateY** Applies a matrix that gives rotation around Y axis.

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateY(*m, ang)
NJS_MATRIX *m
Angle ang
```

### **PARAMETERS**

**NJS\_MATRIX \*m**  
distination matrix

**Angle ang**  
rotation angle

### **RETURN VALUE**

None

### **FUNCTION**

- Applies a matrix that gives rotation around the Y axis to the matrix m.
- When parameter m is NULL, the current matrix will be the destination.

### **EXAMPLE**

The following applies a matrix that gives rotation of 90 degrees around the Y axis to the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njRotateY(NULL, NJM_DEG_ANG(90));
njPopMatrix(1);
```

### **NOTES**

- For a matrix [M] and multiplicative rotational matrix [R],  $[M] = [R] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

### **RELATED TOPICS**

## **njRotateZ**

Applies a matrix that gives rotation around Z axis.

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateZ(*m, ang)
NJS_MATRIX *m
Angle ang
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

destination matrix

**Angle ang**

rotation angle

### **RETURN VALUE**

None

### **FUNCTION**

- Applies a matrix that gives rotation around the Z axis to the matrix m.
- When parameter m is NULL, the current matrix will be the destination.

### **EXAMPLE**

The following applies a matrix that gives rotation of 90 degrees around the Z axis to the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njRotateZ(NULL, NJM_DEG_ANG(90));
njPopMatrix(1);
```

### **NOTES**

- For a matrix [M] and multiplicative rotational matrix [R],  $[M] = [R] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

### **RELATED TOPICS**

## **njRotateXYZ**

Applies a matrix that gives rotation around X, Y, and Z axes.

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateXYZ(*m, ang)
NJS_MATRIX *m
Angle angx
Angle angy
Angle angz
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

destination matrix

**Angle angx**

x rotation angle

**Angle angy**

y rotation angle

**Angle angz**

z rotation angle

### **RETURN VALUE**

None

### **FUNCTION**

- Apply a matrix that gives rotation around the X, Y, and Z axes to the matrix m, in that order.
- When parameter m is NULL, the current matrix will be the destination.

## EXAMPLE

The following applies a matrix that gives rotation of 30 degrees around the X axis, 60 degrees around the Y axis, and 90 degrees around the Z axis to the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njRotateXYZ(NULL, NJM_DEG_ANG(30), NJM_DEG_ANG(60), NJM_DEG_ANG(90));
njPopMatrix(1);
```

## NOTES

- For a matrix [M] and multiplicative rotational matrix [R],  $[M] = [R] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

## RELATED TOPICS

# njScale

Scales a matrix.

## FORMAT

```
#include <Ninja.h>

void njScale(*m, sx, sy, sz)
NJS_MATRIX *m
Float sx
Float sy
Float sz
```

## PARAMETERS

**NJS\_MATRIX \*m**

calculation matrix

**Float sx**

scaling ratio along the X axis

**Float sy**

scaling ratio along the Y axis

**Float sz**

scaling ratio along the Z axis

## RETURN VALUE

None

## FUNCTION

- Scales arbitrary matrix m.
- When parameter m is NULL, the current matrix is scaled.



## EXAMPLE

The following scales the current matrix by 0.5 along the X axis, 1 along the Y axis, and 2 along the Z axis.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njScale(NULL, 0.5f, 1.f, 2.f);
njPopMatrix(1);
```

## NOTES

- For arbitrary matrix [M] and multiplicative scaling matrix [R],  $[M] = [S] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

## RELATED TOPICS

# njScaleV

Scales a matrix.

## FORMAT

```
#include <Ninja.h>

void njScale(*m, *v)
NJS_MATRIX *m
NJS_VECTOR *v
```

## PARAMETERS

**NJS\_MATRIX \*m**

pointer to the calculation matrix

**NJS\_VECTOR \*v**

scaling ratio along the X, Y, and Z axes

## RETURN VALUE

None

## FUNCTION

- Scales arbitrary matrix m.
- When parameter m is NULL, the current matrix is scaled.

## EXAMPLE

The following scales the current matrix by 0.5 along the X axis, 1 along the Y axis, and 2 along the Z axis.

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_VECTOR v;

v.x = 0.5f;
v.y = 1.f;
v.z = 2.f;
njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njScale(NULL, &v);
njPopMatrix(1);
```

## NOTES

- For arbitrary matrix [M] and multiplicative scaling matrix [R],  $[M] = [S] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

## RELATED TOPICS

## njScalor

Returns the scalar of an arbitrary vector.

---

### FORMAT

```
#include <Ninja.h>

Float njScalor(*v)
NJS_VECTOR *v
```

### PARAMETERS

**NJS\_VECTOR \*v**

arbitrary vector

### RETURN VALUE

Size of vector (scalar)

### FUNCTION

- Returns the scalar of an arbitrary vector.

### EXAMPLE

The following returns the size of vector v(3, 3, 3).

```
NJS_VECTOR v;
Float scalar;

v.x = 3.f;
v.y = 3.f;
v.z = 3.f;

scalar = njScalor(&v)

% Result %
scalar = 5.196152
```

### NOTES

### RELATED TOPICS

## **njScalor2** Returns the square of the scalar of an arbitrary vector.

---

### **FORMAT**

```
#include <Ninja.h>

Float njScalor2(*v)
NJS_VECTOR *v
```

### **PARAMETERS**

**NJS\_VECTOR \*v**

arbitrary vector

### **RETURN VALUE**

Square of the vector size (scalar)

### **FUNCTION**

- Returns the square of the scalar of an arbitrary vector.

### **EXAMPLE**

The following returns the square of the scalar of vector v(3, 3, 3).

```
NJS_VECTOR v;
Float scalor2;

v.x = 3.f;
v.y = 3.f;
v.z = 3.f;

scalor2 = njScalor2(&v)

% Result %
scalor2 = 27
```

### **NOTES**

### **RELATED TOPICS**

# njSetMatrix

Copies an arbitrary matrix.

---

## FORMAT

```
#include <Ninja.h>

void njSetMatrix(*md, *ms)
NJS_MATRIX *md
NJS_MATRIX *ms
```

## PARAMETERS

**NJS\_MATRIX \*md**

copy destination matrix

**NJS\_MATRIX \*ms**

copy source matrix

## RETURN VALUE

## FUNCTION

- Copies arbitrary matrix md to matrix ms.
- When parameter md is NULL, the current matrix is copied.

## EXAMPLE

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njSetMatrix(NULL, _unit_matrix_);
njPopMatrix(1);
```

## NOTES

## RELATED TOPICS

NJS\_MATRIX

# njSubMatrix

Performs matrix subtraction.

---

## FORMAT

```
#include <Ninja.h>

void njSubMatrix(*md, *ms)
NJS_MATRIX *md
NJS_MATRIX *ms
```

## PARAMETERS

**NJS\_MATRIX \*md**

minuend matrix

**NJS\_MATRIX \*ms**

subtrahend matrix

## RETURN VALUE

None

## FUNCTION

- Subtracts one matrix from another. ( $[md] = [md] - [ms]$ )
- When parameter m is NULL, the current matrix is taken as the minuend.

## EXAMPLE

## NOTES

## RELATED TOPICS

NJS\_MATRIX

# njSubVector

Performs vector subtraction.

---

## FORMAT

```
#include <Ninja.h>

void njSubVector(*vd, *vs)
NJS_VECTOR *vd
NJS_VECTOR *vs
```

## PARAMETERS

**NJS\_VECTOR \*vd**

minuend vector

**NJS\_VECTOR \*vs**

subtrahend matrix

## RETURN VALUE

## FUNCTION

- Subtracts one vector from another and stores the result in vector vd.

**vd = vd - vs**

## EXAMPLE

## NOTES

## RELATED TOPICS

NJS\_VECTOR

## njTranslate

Applies a matrix that gives parallel translation along each axis.

---

### FORMAT

```
#include <Ninja.h>

void njTranslate(*m, x, y, z)
NJS_MATRIX *m
Float x
Float y
Float z
```

### PARAMETERS

**NJS\_MATRIX \*m**

destination matrix

**Float x**

amount of translation along the X axis

**Float y**

amount of translation along the Y axis

**Float z**

amount of translation along the Z axis

### RETURN VALUE

### FUNCTION

- Applies a matrix that gives parallel translation along each axis to the matrix m.
- When parameter m is NULL, the current matrix will be the destination.



## EXAMPLE

The following applies a matrix that gives parallel translation of 10 along the X axis, 20 along the Y axis, and 30 along the Z axis to the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njTranslate(NULL, 10.f, 20.f, 30.f);
njPopMatrix(1);
```

## NOTES

- For matrix [M] and matrix translated through multiplication [T],  $[M] = [T] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

## RELATED TOPICS

# **njTranslateV**

Moves a matrix laterally.

---

## **FORMAT**

```
#include <Ninja.h>

void njTranslateV(*m, *v)
NJS_MATRIX *m
NJS_VECTOR *v
```

## **PARAMETERS**

**NJS\_MATRIX \*m**

matrix to be moved

**NJS\_VECTOR \*v**

amount of movement along the various axes

## **RETURN VALUE**

None

## **FUNCTION**

- Moves arbitrary matrix m laterally.
- When parameter m is NULL, the current matrix is moved.

## EXAMPLE

The following moves the current matrix by 10 along the X axis, 20 along the Y axis, and 30 along the Z axis.

```
NJS_MATRIX stack[10];
NJS_VIEW view;
NJS_VECTOR v;

v.x = 10.f;
v.y = 20.f;
v.z = 30.f;
njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njTranslateV(NULL, &v);
njPopMatrix(1);
```

## NOTES

- For matrix [M] and matrix translated through multiplication [T],  $[M] = [T] * [M]$ .
- Care must be taken with the order in which matrices are multiplied.

## RELATED TOPICS

# **njTransposeMatrix**

Transposes a matrix.

---

## **FORMAT**

```
#include <Ninja.h>

void njTransposeMatrix(*m)
NJS_MATRIX *m
```

## **PARAMETERS**

**NJS\_MATRIX \*m**

matrix to be transposed

## **RETURN VALUE**

## **FUNCTION**

- Transposes arbitrary matrix m.
- When parameter m is NULL, the current matrix is transposed.

## **EXAMPLE**

The following transposes the current matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njTransposeMatrix(NULL);
njPopMatrix(1);
```

## **NOTES**

For arbitrary matrix M,  $[M] = [M]'$ .

## **RELATED TOPICS**

## **njUnitMatrix**

Converts an arbitrary matrix to a unit matrix.

---

### **FORMAT**

```
#include <Ninja.h>

void njUnitMatrix(*m)
NJS_MATRIX *m
```

### **PARAMETERS**

**NJS\_MATRIX \*m**

matrix to be unitized

### **RETURN VALUE**

None

### **FUNCTION**

- Converts an arbitrary matrix to a unit matrix.
- When parameter m is NULL, the current matrix is unitized.

### **EXAMPLE**

The following converts the current matrix to a unit matrix.

```
NJS_MATRIX stack[10];
NJS_VIEW view;

njInitMatrix(stack, 10);
njInitView(&view);
njSetView(&view);
njClearMatrix();
njPushMatrix(NULL);
njUnitMatrix(NULL);
njPopMatrix(1);
```

### **NOTES**

### **RELATED TOPICS**

NJS\_MATRIX

## njUnitVector

Converts an arbitrary vector to a unit vector.

---

### FORMAT

```
#include <Ninja.h>

Float njUnitVector(*v)
NJS_VECTOR *v
```

### PARAMETERS

**NJS\_VECTOR \*v**

arbitrary vector

### RETURN VALUE

Size of the original vector. (scalar)

### FUNCTION

- Converts an arbitrary vector to a unit vector.

### EXAMPLE

The following converts vector v(3, 3, 3) to a unit vector.

```
NJS_VECTOR v;
Float scalar;

v.x = 3.f;
v.y = 3.f;
v.z = 3.f;

scalar = njUnitVector(&v)

% result %
v(0.577350, 0.577350, 0.577350)
scalar = 5.196152
```

### NOTES

### RELATED TOPICS



## 3. Collision Functions

---

### Contents

<code>njCollisionCheckBB</code> .....	Checks Collision for 2 Hexahedron. ....	NLS-84
<code>njCollisionCheckBC</code> .....	Checks collision for a hexahedron and a capsule. ....	NLS-86
<code>njCollisionCheckBS</code> .....	Checking for collision for a hexahedron and a sphere. ....	NLS-88
<code>njCollisionCheckCC</code> .....	Checks collision for two capsules. ....	NLS-90
<code>njCollisionCheckSC</code> .....	Checks collision for a sphere and a capsule. ....	NLS-92
<code>njCollisionCheckSS</code> .....	Checks collision for two spheres. ....	NLS-94
<code>njDistanceL2L</code> .....	Returns the distance between two lines. ....	NLS-96
<code>njDistanceL2PL</code> .....	Returns the distance between a line and a plane. ....	NLS-97
<code>njDistanceP2L</code> .....	Returns the distance between a point and a line. ....	NLS-99
<code>njDistanceP2P</code> .....	Returns the distance between two points. ....	NLS-101
<code>njDistanceP2PL</code> .....	Returns the distance between a point and a line. ....	NLS-102
<code>njDistancePL2PL</code> .....	Returns the distance between two planes. ....	NLS-104
<code>njGetPnaneNormal</code> .....	Finds the vector that is normal to a plane. ....	NLS-105
<code>njGetPnaneNormal2</code> .....	Finds the vector that is normal to a plane. ....	NLS-107
<code>njIsParalellL2L</code> .....	Returns whether two lines are parallel. ....	NLS-109
<code>njIsParalellL2PL</code> .....	Returns whether a line and a plane are parallel. ....	NLS-111
<code>njIsParalellPL2PL</code> .....	Returns whether two planes are parallel. ....	NLS-113

## **njCollisionCheckBB**

Checks Collision for 2 Hexahedron.

---

### **FORMAT**

```
#include <Ninja.h>
Int njCollisionCheckBB(*h1, *h2)
NJS_BOX *h1
NJS_BOX *h2
```

### **PARAMETERS**

**\*h1**

Hexahedron object #1 to be collision checked

**\*h2**

Hexahedron object #2 to be collision checked

### **RETURN VALUE**

1: Hit

0: Not hit

### **ERROR VALUE**

None

### **FUNCTION**

Checks collision for two hexahedrons.

### **EXAMPLE**

Check collision for two hexahedron h1 and h2:

```
NJS_BOX h1, h2;
```

```
h1.v[0].x = -150.f; h1.v[0].y = 100.f; h1.v[0].z = 100;
h1.v[1].x = -150.f; h1.v[1].y = 100.f; h1.v[1].z = -100;
h1.v[2].x = 50.f; h1.v[2].y = 100.f; h1.v[2].z = -100;
h1.v[3].x = 50.f; h1.v[3].y = 100.f; h1.v[3].z = 100;
h1.v[4].x = -150.f; h1.v[4].y = -100.f; h1.v[4].z = 100;
h1.v[5].x = -150.f; h1.v[5].y = -100.f; h1.v[5].z = -100;
h1.v[6].x = 50.f; h1.v[6].y = -100.f; h1.v[6].z = -100;
h1.v[7].x = 50.f; h1.v[7].y = -100.f; h1.v[7].z = 100;
```



```
h2.v[0].x = -50.f; h2.v[0].y = 100.f; h2.v[0].z = 100;
h2.v[1].x = -50.f; h2.v[1].y = 100.f; h2.v[1].z = -100;
h2.v[2].x = 150.f; h2.v[2].y = 100.f; h2.v[2].z = -100;
h2.v[3].x = 150.f; h2.v[3].y = 100.f; h2.v[3].z = 100;
h2.v[4].x = -50.f; h2.v[4].y = -100.f; h2.v[4].z = 100;
h2.v[5].x = -50.f; h2.v[5].y = -100.f; h2.v[5].z = -100;
h2.v[6].x = 150.f; h2.v[6].y = -100.f; h2.v[6].z = -100;
h2.v[7].x = 150.f; h2.v[7].y = -100.f; h2.v[7].z = 100;

njCollisionCheckBB(&h1, &h2);
```

**Result**

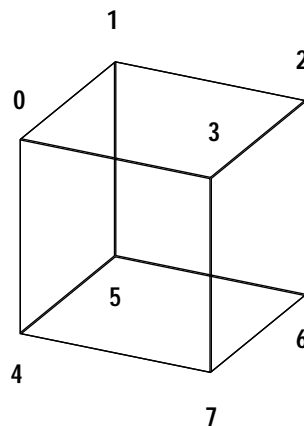
Return value: 1

## NOTES

**NJS\_BOX structure (box type)**

```
typedef struct{
                                NJS_POINT3 v[8]; /* hexahedron vertex list */
} NJS_BOX;
```

Corresponding vertices with the vertex list index



## RELATED TOPICS

## **njCollisionCheckBC**

Checks collision for a hexahedron and a capsule.

---

### **FORMAT**

```
#include <Ninja.h>
Int njCollisionCheckBC( *box, *capsule )
NJS_BOX      *box
NJS_CAPSULE  *capsule
```

### **PARAMETERS**

#### **\*box**

Hexahedron to be collision checked

#### **\*capsule**

Capsule to be collision checked

### **RETURN VALUE**

1: Hit

0: Not hit

### **ERROR VALUE**

None

### **FUNCTION**

Checks collision for a hexahedron and a capsule

### **EXAMPLE**

Hexahedron (box) and capsule (capsule) collision checking.

```
NJS_BOX box;
NJS_CAPSULE capsule;

box.v[0].x = -150.f; box.v[0].y = 100.f; box.v[0].z = 100;
box.v[1].x = -150.f; box.v[1].y = 100.f; box.v[1].z = -100;
box.v[2].x = 50.f; box.v[2].y = 100.f; box.v[2].z = -100;
box.v[3].x = 50.f; box.v[3].y = 100.f; box.v[3].z = 100;
box.v[4].x = -150.f; box.v[4].y = -100.f; box.v[4].z = 100;
box.v[5].x = -150.f; box.v[5].y = -100.f; box.v[5].z = -100;
box.v[6].x = 50.f; box.v[6].y = -100.f; box.v[6].z = -100;
box.v[7].x = 50.f; box.v[7].y = -100.f; box.v[7].z = 100;
```

```
capsule.c1.x = 100.f;
capsule.c1.y = 0.f;
capsule.c1.z = 0.f;
capsule.c2.x = 200.f;
capsule.c2.y = 0.f;
capsule.c2.z = 0.f;
capsule.r = 100.f;

njCollisionCheckBC(&box, &capsule);
```

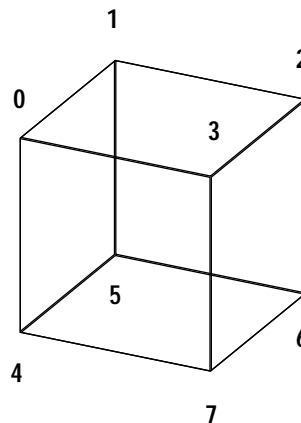
**Result**

Return value: 1

**NOTES****RELATED TOPICS****NJS\_BOX structure(box type)**

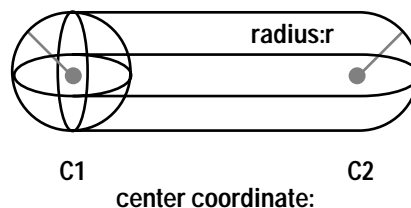
```
typedef struct{
    NJS_POINT3 v[8]; /* vertex list for hexahedron */
} NJS_BOX;
```

Corresponding vertices with vertex list index

**NJS\_CAPSULE structure(capsule type)**

```
typedef struct{
    NJS_POINT3 c1;
    NJS_POINT3 c2;
    Float      r;
} NJS_CAPSULE;
```

A capsule consists of a half sphere with center coordinate = c1 and radius = r one side, and another half sphere with center coordinate = c2 and radius = r the other side.



## njCollisionCheckBS

Checking for collision for a hexahedron and a sphere.

---

### FORMAT

```
#include <Ninja.h>
Int njCollisionCheckBS( *box, *sphere )
NJS_BOX *box
NJS_SPHERE *sphere
```

### PARAMETERS

**\*box**

Hexahedron to be collision checked

**\*sphere**

Sphere to be collision checked

### RETURN VALUE

1: Hit

0: Not hit

### ERROR VALUE

None

### FUNCTION

Checks collision for a hexahedron and a sphere.

### EXAMPLE

Hexahedron (box) and sphere (sphere) collision checking.

```
NJS_BOX box;
NJS_SPHERE sphere;
```

```
box.v[0].x = -150.f; box.v[0].y = 100.f; box.v[0].z = 100;
box.v[1].x = -150.f; box.v[1].y = 100.f; box.v[1].z = -100;
box.v[2].x = 50.f; box.v[2].y = 100.f; box.v[2].z = -100;
box.v[3].x = 50.f; box.v[3].y = 100.f; box.v[3].z = 100;
box.v[4].x = -150.f; box.v[4].y = -100.f; box.v[4].z = 100;
box.v[5].x = -150.f; box.v[5].y = -100.f; box.v[5].z = -100;
box.v[6].x = 50.f; box.v[6].y = -100.f; box.v[6].z = -100;
box.v[7].x = 50.f; box.v[7].y = -100.f; box.v[7].z = 100;
```

```
sphere.c.x = 100.f;  
sphere.c.y =  0.f;  
sphere.c.z =  0.f;  
sphere.r   = 100.f;  
  
njCollisionCheckBS(&box, &sphere);
```

**Result**

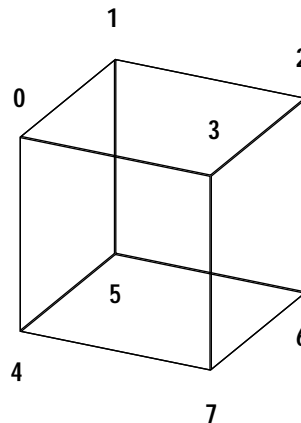
Return value: 1

## NOTES

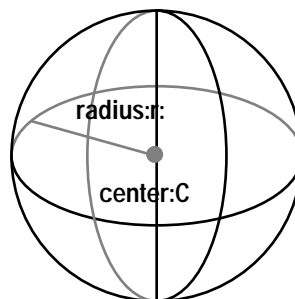
**NJS\_BOX structure(box type)**

```
typedef struct{  
    NJS_POINT3 v[8]; /* vertex list for hexahedron */  
} NJS_BOX;
```

Corresponding vertices with vertex list index

**NJS\_SPHERE structure(sphere)**

```
typedef struct{  
    NJS_POINT3 c; /* center of the sphere */  
    Float      r; /* radius of the sphere */  
} NJS_SPHERE;
```



A sphere consists of center coordinate c and radius r

## RELATED TOPICS

## **njCollisionCheckCC**

Checks collision for two capsules.

---

### **FORMAT**

```
#include <Ninja.h>
Int njCollisionCheckCC( *h1, *h2 )
NJS_CAPSULE *h1
NJS_CAPSULE *h2
```

### **PARAMETERS**

**\*h1**

Collision checked object capsule #1

**\*h2**

Collision checked object capsule #2

### **RETURN VALUE**

1: Hit

0: Not hit

### **ERROR VALUE**

None

### **FUNCTION**

Checks collision for two capsules.

## EXAMPLE

Check collision for two capsules h1 and h2.

```
NJS_CAPSULE h1,h2;
```

```
h1.c1.x = -80.f;  
h1.c1.y = 0.f;  
h1.c1.z = 0.f;  
h1.c2.x = -180.f;  
h1.c2.y = 0.f;  
h1.c2.z = 0.f;  
h1.r = 100.f;
```

```
h2.c1.x = 80.f;  
h2.c1.y = 0.f;  
h2.c1.z = 0.f;  
h2.c2.x = 180.f;  
h2.c2.y = 0.f;  
h2.c2.z = 0.f;  
h2.r = 100.f;
```

```
njCollisionCheckCC(&h1, &h2);
```

### Result

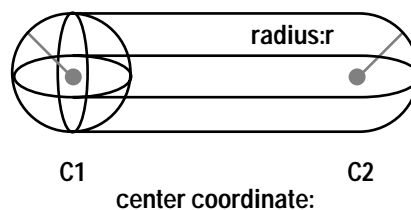
Return value: 1

## NOTES

### NJS\_CAPSULE structure (capsule type)

```
typedef struct{  
    NJS_POINT3 c1;  
    NJS_POINT3 c2;  
    Float      r;  
} NJS_CAPSULE;
```

A capsule consists of a half sphere with center coordinate = c1 and radius = r on one side, and another half sphere with center coordinate = c2 and radius = r on the other side.



## RELATED TOPICS

## **njCollisionCheckSC**

Checks collision for a sphere and a capsule.

---

### **FORMAT**

```
#include <Ninja.h>
Int njCollisionCheckSC( *sphere, *capsule)
NJS_SPHERE *sphere
NJS_CAPSULE *capsule
```

### **PARAMETERS**

#### **\*sphere**

Sphere to be collision checked

#### **\*capsule**

Capsule to be collision checked

### **RETURN VALUE**

1: Hit

0: Not hit

### **ERROR VALUE**

None

### **FUNCTION**

Checks collision for a sphere and a capsule.

### **EXAMPLE**

Checks collision for a sphere and a capsule.

```
NJS_SPHERE *sphere
NJS_CAPSULE *capsule

sphere.c.x = -80.f;
sphere.c.y = 0.f;
sphere.c.z = 0.f;
sphere.r = 100.f;

capsule.cl.x = 80.f;
capsule.cl.y = 0.f;
```



```
capsule.c1.z = 0.f;
capsule.c2.x = 180.f;
capsule.c2.y = 0.f;
capsule.c2.z = 0.f;
capsule.r = 100.f;

njCollisionCheckSC(&sphere, &capsule);
```

**Result**

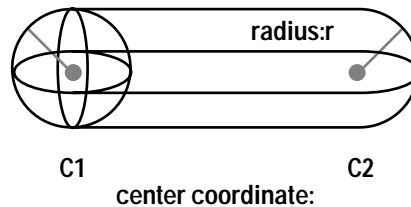
Return value: 1

## NOTES

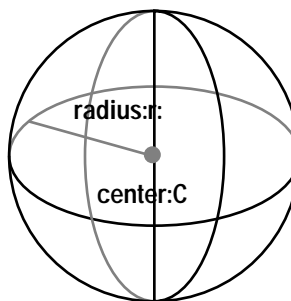
**NJS\_CAPSULE structure (capsule type)**

```
typedef struct{
    NJS_POINT3 c1;
    NJS_POINT3 c2;
    Float      r;
} NJS_CAPSULE;
```

A capsule consisted of a half sphere with center coordinate = c1 and radius = r one side, and another half sphere with center coordinate = c2 and radius = r the other side.

**NJS\_SPHEREstructure (sphere)**

```
typedef struct{
    NJS_POINT3 c; /* center of sphere */
    Float      r; /* radius of sphere */
} NJS_SPHERE;
```



A sphere consists of center coordinate c and radius r.

## RELATED TOPICS

## **njCollisionCheckSS**

Checks collision for two spheres.

---

### **FORMAT**

```
#include <Ninja.h>
Int njCollisionCheckSS( *sphere1, *sphere2 )
NJS_SPHERE *sphere1
NJS_SPHERE *sphere2
```

### **PARAMETERS**

**\*sphere1**

Collision checked object, sphere #1

**\*sphere2**

Collision checked object, sphere #2

### **RETURN VALUE**

1: Hit

0: Not hit

### **ERROR VALUE**

None

### **FUNCTION**

Checks collision for two spheres.

## EXAMPLE

Checks collision for two spheres, sphere1 and sphere2.

```
NJS_SPHERE sphere1,sphere2;
```

```
sphere1.c.x = -80.f;  
sphere1.c.y = 0.f;  
sphere1.c.z = 0.f;  
sphere1.r = 100.f;  
sphere2.c.x = 80.f;  
sphere2.c.y = 0.f;  
sphere2.c.z = 0.f;  
sphere2.r = 100.f;
```

```
njCollisionCheckSS(&sphere1, &sphere2);
```

### Result

Return value: 1

## NOTES

### NJS\_SPHEREstructure (sphere)

```
typedef struct{
```

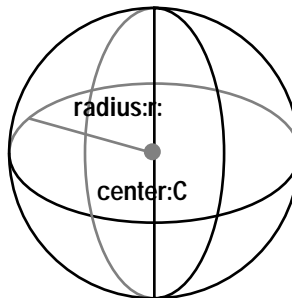
```
    NJS_POINT3
```

```
    Float
```

```
    c; /* center of sphere */
```

```
    r; /* radius of sphere */
```

```
} NJS_SPHERE;
```



A sphere consists of center coordinate c and radius r.

## RELATED TOPICS

## **njDistanceL2L**

Returns the distance between two lines.

---

### **FORMAT**

```
#include <Ninja.h>

Float njDisatnceL2L(*l1, *l2, *p1, *p2)
NJS_LINE *l1
NJS_LINE *l2
NJS_POINT3 *p1
NJS_POINT3 *p2
```

### **PARAMETERS**

**NJS\_LINE \*l1**

line data 1

**NJS\_LINE \*l2**

line data 2

**NJS\_POINT3 \*p1**

coordinates of closest point on line 1

**NJS\_POINT3 \*p2**

coordinates of closest point on line 2

### **RETURN VALUE**

Minimum distance between the two lines.

### **FUNCTION**

- Returns the minimum distance between two lines.
- When parameters p1 and 2 are not NULL, the coordinates of the two closest points on the two lines are determined and stored in p1 (for the point on l1) and p2 (for the point on l2).

### **EXAMPLE**

### **NOTES**

### **RELATED TOPICS**

NJS\_POINT3

NJS\_LINE

## **njDistanceL2PL**      Returns the distance between a line and a plane.

---

### **FORMAT**

```
#include <Ninja.h>

Float njDistanceL2L(*l, *pl, *cp)
NJS_LINE    *l
NJS_PLANE    *pl
NJS_POINT3   *cp
```

### **PARAMETERS**

**NJS\_LINE \*l**

line data

**NJS\_PLANE \*pl**

plane data

**NJS\_POINT3 \*cp**

intersection of a line and a plane (if not parallel)

### **RETURN VALUE**

Distance between a line and a plane.

### **FUNCTION**

- Returns the distance between a line and a plane.
- When parameter cp is not NULL, the coordinates of the point of intersection between the line and the plane are calculated and stored in cp (when the line and plane are not parallel).
- When the line and plane are parallel, the coordinates of the point of intersection between a projection from the start of the line, to the plane, is stored.

### EXAMPLE

```
NJS_LINE    l;  
NJS_PLANE   pl;  
NJS_POINT3  cp;  
  
l.px = 2.f;  
l.py = -1.f;  
l.pz = 3.f;  
l.vx = 4.f;  
l.vy = -1.f;  
l.vz = 1.f;  
pl.px = 0.f;  
pl.py = 0.f;  
pl.pz = 0.f;  
pl.vx = 1.f;  
pl.vy = -4.f;  
pl.vz = 1.f;  
  
njDistanceL2PL(&l, &pl, &cp);  
  
% Result %  
Return value 0.000000  
cp (-2, 0, 2)
```

### NOTES

### RELATED TOPICS

NJS\_POINT3

NJS\_LINE

NJS\_PLANE

## **njDistanceP2L**

Returns the distance between a point and a line.

---

### **FORMAT**

```
#include <Ninja.h>

Float njDistanceP2L(*p, *l, *cp)
NJS_POINT3 *p
NJS_LINE *l
NJS_POINT3 *cp
```

### **PARAMETERS**

**NJS\_POINT3 \*p**

point data

**NJS\_LINE \*l**

line data

**NJS\_POINT3 \*cp**

coordinates of the base of the projection from the point p to the line l

### **RETURN VALUE**

Distance between a point and a line.

### **FUNCTION**

- Returns the distance between a point and a line.
- When parameter cp is not NULL, the coordinates of the base of the projection from the point p to the line l are returned in cp.

### EXAMPLE

```
NJS_POINT3 p, cp;
NJS_LINE l;

p.x = 6.f;
p.y = 6.f;
p.z = 4.f;
l.px = 4.f;
l.py = 3.f;
l.pz = 2.f;
l.vx = 3.f;

l.vy = 2.f;
l.vz = 1.f;

njDistanceP2L(&p, &l, &cp);

% Result %
Return value  1.732051
cp (7, 5, 3)
```

### NOTES

### RELATED TOPICS

NJS\_POINT3 NJS\_LINE



## njDistanceP2P

Returns the distance between two points.

---

### FORMAT

```
#include <Ninja.h>

Float njDistanceP2P(*p1, *p2)
NJS_POINT3 *p1
NJS_POINT3 *p2
```

### PARAMETERS

**NJS\_POINT3 \*p1**

point data 1

**NJS\_POINT3 \*p2**

point data 2

### RETURN VALUE

Distance between the two points.

### FUNCTION

- Returns the distance between two points.

### EXAMPLE

```
NJS_POINT3 p1, p2;

p1.x = 100.f;
p1.y = 100.f;
p1.z = 100.f;
p2.x = -100.f;
p2.y = -100.f;
p2.z = -100.f;

njDistanceP2P(&p1, &p2);

% Result %
346.410156
```

### NOTES

### RELATED TOPICS

NJS\_POINT3

## **njDistanceP2PL** Returns the distance between a point and a line.

---

### **FORMAT**

```
#include <Ninja.h>

Float njDistanceP2PL(*p, *pl, *cp)
NJS_POINT3 *p
NJS_PLANE *pl
NJS_POINT3 *cp
```

### **PARAMETERS**

**NJS\_POINT3 \*p**

point data

**NJS\_PLANE \*pl**

line data

**NJS\_POINT3 \*cp**

coordinates of the base of the projection from the point p to the line l

### **RETURN VALUE**

Distance between a point and a line

### **FUNCTION**

- Returns the distance between a point and a line.
- When parameter cp is not NULL, the coordinates of the base of the projection from the point p to the line l are returned in cp.

## EXAMPLE

```
NJS_POINT3 p, cp;
NJS_PLANE pl;

p.x = 100.f;
p.y = 100.f;
p.z = 100.f;
pl.px = 0.f;
pl.py = 0.f;
pl.pz = 0.f;
pl.vx = 0.f;
pl.vy = 0.f;
pl.vz = 1.f;

njDistanceP2PL(&p, &pl, &cp);

% Result %
Return value 100.000000
cp (100, 100, 0)
```

## NOTES

## RELATED TOPICS

NJS\_POINT3

NJS\_LINE

## **njDistancePL2PL**

Returns the distance between two planes.

---

### **FORMAT**

```
#include <Ninja.h>

Float njDisatnceL2L(*p11, *p12, *l)
NJS_PLANE  *p11
NJS_PLANE  *p12
NJS_LINE   *l
```

### **PARAMETERS**

**NJS\_PLANE \*p11**

plane data 1

**NJS\_PLANE \*p12**

plane data 2

**NJS\_LINE \*l**

line of intersection between the two planes

### **RETURN VALUE**

Distance between the two planes

### **FUNCTION**

- Returns the distance between two planes.
- When parameter l is not NULL, the line of intersection between the two planes is determined and stored in l (if the two planes are not parallel).
- If the two planes are parallel, the value of the members of l is 0.

### **EXAMPLE**

### **NOTES**

### **RELATED TOPICS**

NJS\_LINE

NJS\_PLANE

## **njGetPnaneNormal**

Finds the vector that is normal to a plane.

---

### **FORMAT**

```
#include <Ninja.h>
void njGetPlaneNormal(*p, *v)
NJS_POINT3 *p
NJS_VECTOR *v
```

### **PARAMETERS**

**NJS\_POINT3 \*p**

data array describing three points in the plane

**NJS\_VECTOR \*v**

normal vector to the plane

### **RETURN VALUE**

None

### **FUNCTION**

- Finds the vector that is normal to a plane.

### EXAMPLE

The following finds the vector that is normal to the plane passing through points (1, 2, 3), (1, -1, 2), and (2, 3, 1).

```
NJS_POINT3 p[3];
NJS_VECTOR v;

p[0].x = 1.f;
p[0].y = 2.f;
p[0].z = 3.f;
p[1].x = 1.f;
p[1].y = -1.f;
p[1].z = 2.f;
p[2].x = 2.f;
p[2].y = 3.f;
p[2].z = 1.f;

njGetPlaneNormal(p, &v);

% Result %
v (7, -1, 3)
```

### NOTES

### RELATED TOPICS

NJS\_POINT3

NJS\_VECTOR

## **njGetPnaneNormal2**

Finds the vector that is normal to a plane.

---

### **FORMAT**

```
#include <Ninja.h>
void njGetPlaneNormal(*p0, *p1, *p2 *v)
NJS_POINT3 *p0
NJS_POINT3 *p1
NJS_POINT3 *p2
NJS_VECTOR *v
```

### **PARAMETERS**

**NJS\_POINT3 \*p0**

data of point 1 on plane

**NJS\_POINT3 \*p1**

data of point 2 on plane

**NJS\_POINT3 \*p2**

data of point 3 on plane

**NJS\_VECTOR \*v**

normal vector to plane

### **RETURN VALUE**

None

### **FUNCTION**

- Finds the normal vector from three points on a plane.

### EXAMPLE

The following finds the normal vector to the plane passing through points (1, 2, 3), (1, -1, 2), and (2, 3, 1).

```
NJS_POINT3 p0, p1, p2;
NJS_VECTOR v;

p0.x = 1.f;
p0.y = 2.f;
p0.z = 3.f;
p1.x = 1.f;
p1.y = -1.f;
p1.z = 2.f;
p2.x = 2.f;
p2.y = 3.f;
p2.z = 1.f;

njGetPlaneNormal(&p0, &p1, &p2, &v);

% Result %
v (7, -1, 3)
```

### NOTES

### RELATED TOPICS

NJS\_POINT3  
NJS\_VECTOR



## **njIsParalellL2L**

Returns whether two lines are parallel.

---

### **FORMAT**

```
#include <Ninja.h>

Bool njIsParalellL2L(*l1, *l2)
NJS_LINE *l1
NJS_LINE *l2
```

### **PARAMETERS**

**NJS\_LINE \*l1**

line 1 data

**NJS\_LINE \*l2**

line 2 data

### **RETURN VALUE**

**TRUE**

parallel

**FALSE**

not parallel

### **FUNCTION**

- Determines whether or not two lines are parallel.

### EXAMPLE

```
NJS_LINE l1, l2;

l1.px = 100.f;
l1.py = 100.f;
l1.pz = 100.f;
l1.vx = 3.f;
l1.vy = 4.f;
l1.vz = 5.f;

l2.px = 100.f;
l2.py = 100.f;
l2.pz = -100.f;
l2.vx = -3.f;
l2.vy = -4.f;
l2.vz = -5.f;

njIsParalellL2L(&l1, &l2);

% Result %
TRUE
```

### NOTES

### RELATED TOPICS

NJS\_LINE

## **njIsParalellL2PL**

Returns whether a line and a plane are parallel.

---

### **FORMAT**

```
#include <Ninja.h>

Bool njIsParalellL2L(*l, *pl)
NJS_LINE   *l
NJS_PLANE  *pl
```

### **PARAMETERS**

**NJS\_LINE \*l**

line data

**NJS\_PLANE \*pl**

plane data

### **RETURN VALUE**

**TRUE**

parallel

**FALSE**

not parallel

### **FUNCTION**

- Determines whether or not a line and a plane are parallel.

### EXAMPLE

```
NJS_LINE l;  
NJS_PLANE pl;  
  
l.px = 100.f;  
l.py = 100.f;  
l.pz = 100.f;  
l.vx = 1.f;  
l.vy = 0.f;  
l.vz = 0.f;  
  
pl.px = -100.f;  
pl.py = -100.f;  
pl.pz = -100.f;  
pl.vx = -0.f;  
pl.vy = -0.f;  
pl.vz = -1.f;  
  
njIsParalellL2PL(&l, &pl);  
  
% Result %  
TRUE
```

### NOTES

### RELATED TOPICS

NJS\_LINE  
NJS\_PLANE

## **njIsParalelPL2PL**

Returns whether two planes are parallel.

---

### **FORMAT**

```
#include <Ninja.h>

Bool njIsParalelPL2PL(*p11, *p12)
NJS_PLANE *p11
NJS_PLANE *p12
```

### **PARAMETERS**

**NJS\_PLANE \*p11**

plane 1 data

**NJS\_PLANE \*p12**

plane 2 data

### **RETURN VALUE**

**TRUE**

parallel

**FALSE**

not parallel

### **FUNCTION**

Determines whether or not two planes are parallel.

### EXAMPLE

```
NJS_PLANE pl1, pl2;

pl1.px = 100.f;
pl1.py = 100.f;
pl1.pz = 100.f;
pl1.vx = 0.f;
pl1.vy = 0.f;
pl1.vz = 1.f;

pl2.px = -100.f;
pl2.py = -100.f;
pl2.pz = -100.f;
pl2.vx = 0.f;
pl2.vy = 0.f;
pl2.vz = -1.f;

njIsParalellPL2PL(&pl1, &pl2);

% Result %
TRUE
```

### NOTES

### RELATED TOPICS

NJS\_PLANE



## 4. *Mathmmatical Functions*

---

### Contents

njAbs.....	Returns an absolute value. ....	NLS-117
njArcCos.....	Returns an arc cosine (ArcCos). ....	NLS-118
njArcCosec.....	Returns an arc cosecant. ....	NLS-119
njArcCot.....	Returns an arc cotangent. ....	NLS-120
njArcSec.....	Returns an arc secant. ....	NLS-121
njArcSin.....	Returns an arc sine. ....	NLS-122
njArcTan.....	Returns an arc tangent. ....	NLS-123
njArcTan2.....	Returns an arc tangent (ArcTan2). ....	NLS-124
njBezier.....	<b>Bezier correction function</b> .....	NLS-125
njCardinalSpline.....	<b>Cardinal spline correction function</b> .....	NLS-127
njCeil.....	Returns the smallest integer not less than n (Ceiling Function). ....	NLS-129
njCombination.....	<b>Combination function</b> .....	NLS-130
njCos.....	Returns a cosine. ....	NLS-131
njCosec.....	Returns the cosecant. ....	NLS-132
njCosech.....	Returns the hyperbolic cosecant. ....	NLS-133
njCosh.....	Returns the hyperbolic cosine. ....	NLS-134
njCot.....	Returns the cotangent. ....	NLS-135
njCoth.....	Returns the hyperbolic cotangent. ....	NLS-136
njCubicBezier.....	<b>3D Bezier spline correction function</b> .....	NLS-137
njExp.....	Returns exponents. ....	NLS-139
njFloor.....	Returns the largest integer not greater than n (Floor Function). ....	NLS-140
njFraction.....	Returns the decimal fraction. ....	NLS-141
njHermite.....	<b>Hermite spline correction function</b> .....	NLS-142
njHypot.....	Returns length of a hypotenuse. ....	NLS-144
njInvertSqrt.....	Returns the inverse square root. ....	NLS-145
njKochanekSpline.....	<b>Kochanek spline correction function</b> .....	NLS-146
njLinear.....	<b>Line shape correction function</b> .....	NLS-148

njLog.....	Returns the natural logarithm. ....	NLS-150
njLog10.....	Returns the base 10 logarithm. ....	NLS-151
njLog2.....	Returns the base 2 logarithm. ....	NLS-152
njOverhauserSpline .....	<b>Overhauser spline correction function</b> .....	<b>NLS-153</b>
njPow .....	Returns the power of a number. ....	NLS-155
njRandom .....	Generates a random number. ....	NLS-156
njRandomSeed .....	Sets the random number seed. ....	NLS-157
njRoundOff.....	Rounds down the decimal fraction. ....	NLS-158
njRoundUp .....	Rounds up the decimal fraction. ....	NLS-159
njSec .....	Returns a secant. ....	NLS-160
njSech.....	Returns a hyperbolic secant. ....	NLS-161
njSin .....	Returns a sine. ....	NLS-162
njSinh.....	Returns a hyperbolic sine. ....	NLS-163
njSqrt.....	Returns a square root. ....	NLS-164
njTan .....	Returns a tangent. ....	NLS-165
njTanh.....	Returns a hyperbolic tangent. ....	NLS-166



## njAbs

Returns an absolute value.

---

### FORMAT

```
#include <Ninja.h>
Float njAbs( n )
Float n
```

### PARAMETERS

**n**

arbitrary real number

### RETURN VALUE

Absolute value of real number n

### FUNCTION

- Determines the absolute value of arbitrary real number n and returns that value as the result.

### EXAMPLE

```
int ix = -4, iy;
double dx = -3.141593, dy;
iy = njAbs( ix );
dy = njAbs( dx );
% Result %
The absolute value of -4 is 4.
The absolute value of -3.141593 is 3.141593.
```

### NOTES

### RELATED TOPICS

## **njArcCos**

Returns an arc cosine (ArcCos).

---

### **FORMAT**

```
#include <Ninja.h>
Angle njArcCos( n )
Float n
```

### **PARAMETERS**

**n**

value of cosine

### **RETURN VALUE**

value of arc cosine

### **FUNCTION**

- Obtains the arc cosine of arbitrary cosine n and returns the result.

### **EXAMPLE**

```
ans = njArcCos( 0.5f );
% Result %
ans = 2aaa
```

### **NOTES**

### **RELATED TOPICS**

[njArcCosec](#)

[njArcCot](#)

[njArcSec](#)

[njArcSin](#)

[njArcTan](#)

## **njArcCosec**

Returns an arc cosecant.

---

### **FORMAT**

```
#include <Ninja.h>
Angle njArcCosec( n )
Float n
```

### **PARAMETERS**

**n**

arbitrary real number

### **RETURN VALUE**

Value of arc cosecant

### **FUNCTION**

- Determines the arc cosecant of arbitrary real number n and returns the value found as the result.

### **EXAMPLE**

```
ans = njArcCosec( 2.f );
% Result %
ans = 1555
```

### **NOTES**

### **RELATED TOPICS**

[njArcCos](#)

[njArcCot](#)

[njArcSec](#)

[njArcSin](#)

[njArcTan](#)

# **njArcCot**

Returns an arc cotangent.

---

## **FORMAT**

```
#include <Ninja.h>
Angle njArcCot( n )
Float n
```

## **PARAMETERS**

**n**

arbitrary real number

## **RETURN VALUE**

Value of arc cotangent

## **FUNCTION**

- Determines the arc cotangent of arbitrary real number n and returns the value found as the result.

## **EXAMPLE**

```
ans = njArcCot( 2.f );
% Result %
ans = 12e4
```

## **NOTES**

## **RELATED TOPICS**

[njArcCos](#)

[njArcCosec](#)

[njArcSec](#)

[njArcSin](#)

[njArcTan](#)

# **njArcSec**

Returns an arc secant.

---

## **FORMAT**

```
#include <Ninja.h>
Angle njArcSec( n )
Float n
```

## **PARAMETERS**

**n**

an arbitrary real number

## **RETURN VALUE**

Value of arc secant

## **FUNCTION**

- Determines the arc secant of arbitrary real number n and returns the value found as the result.

### **EXAMPLE**

```
ans = njArcSec( 2.f );
% Result %
ans = 2aaa
```

## **NOTES**

## **RELATED TOPICS**

[njArcCos](#)

[njArcCosec](#)

[njArcCot](#)

[njArcSin](#)

[njArcTan](#)

# **njArcSin**

Returns an arc sine.

---

## **FORMAT**

```
#include <Ninja.h>
Angle njArcSin( n )
Float n
```

## **PARAMETERS**

**n**

value of sine

## **RETURN VALUE**

Value of arc sine

## **FUNCTION**

- Determines the arc sine of an arbitrary given sine and returns the value found as the result.

## **EXAMPLE**

```
ans = njArcSin( 0.5f );
% Result %
ans = 1555
```

## **NOTES**

## **RELATED TOPICS**

[njArcCos](#)

[njArcCosec](#)

[njArcCot](#)

[njArcSec](#)

[njArcTan](#)

# njArcTan

Returns an arc tangent.

---

## FORMAT

```
#include <Ninja.h>
Angle njArcTan( n )
Float n
```

## PARAMETERS

**n**

value of tangent

## RETURN VALUE

Value of arc tangent

## FUNCTION

- Determines the arc tangent of a given arbitrary tangent value n and returns the value found as the result.

## EXAMPLE

```
ans = njArcTan( 2.f );
% Result %
ans = 2d1b
```

## NOTES

## RELATED TOPICS

[njArcCos](#)

[njArcCosec](#)

[njArcCot](#)

[njArcSec](#)

[njArcSin](#)

## njArcTan2

Returns an arc tangent (ArcTan2).

---

### FORMAT

```
#include <Ninja.h>
Angle njArcTan2( y, x )
Float x
Float y
```

### PARAMETERS

**x**

length of base

**y**

altitude

### RETURN VALUE

Value of arc tangent

### FUNCTION

- Determines the arc tangent from a given base x and altitude y and returns the value of the result.

### EXAMPLE

```
ans = njArcTan2( 4.f , 2.f );
% Result %
ans = 2d1b
```

### NOTES

### RELATED TOPICS

[njArcCos](#)

[njArcCosec](#)

[njArcCot](#)

[njArcSec](#)

[njArcSin](#)



## njBezier

## Bezier correction function

---

### Syntax

```
#include <Ninja.h>
void njBezier(float *idata, float *odata, NJS_SPLINE *attr, float frame);
float      *idata
float      *odata
NJS_SPLINE *attr
float      frame
```

### Arguments

**idata**

Input data (but idata[3\*sample point number])

**odata**

Output data (but odata[3])

**attr**

Spline attribute

**frame**

Frame

### Return Value

None

### Error

None

### Description

Bezier curve correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.

For the attributes, please specify the sample point number  
at the iparam member (attr->iparam[0]) of the spline attribute structure.

### Usage Example

```
#include <NINJA.H>
#define X1      1.f
#define Y1      0.f
#define Z1      0.f
#define X2      5.f
    .....
#define POINTS  5
float idata[3][5] =
{
  {X1, Y1, Z1},
  {X2, Y2, Z2},
  {X3, Y3, Z3},
  {X4, Y4, Z4},
  {X5, Y5, Z5},
};
float odata[3];
NJS_SPLINE  attr;
    .....
attr.iparam[0] = POINTS;
njBezier((float*)idata, odata, &attr, 0.5f);
    .....
```

### Remark

None

### Related Items

`njCubicBezier`

## njCardinalSpline

## Cardinal spline correction function

---

### Syntax

```
#include <Ninja.h>
void  njCardinalSpline(float *idata, float *odata, NJS_SPLINE *attr, float frame);
float      *idata
float      *odata
NJS_SPLINE *attr
float      frame
```

### Arguments

**idata**

Input data (but idata[3\*4])

**odata**

Output data (but odata[3])

**attr**

Spline attribute

**frame**

Frame

### Return Value

None

### Error

None

### Description

Spline curve correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.

For the attributes, please specify tension at the fparam member (attr->fparam[0]) of the spline attribute structure.

## Usage Example

```
#include <NINJA.H>
#define X1  1.f
#define Y1  0.f
#define Z1  0.f
#define X2  5.f
#define TENSION 0.2f

.....

float idata[3][4] =
{
  {X1, Y1, Z1},
  {X2, Y2, Z2},
  {X3, Y3, Z3},
  {X4, Y4, Z4},
};
float odata[3];
NJS_SPLINE attr;

.....

attr.fparam[0] = TENSION;
njCardinalSpline((float*)idata, odata, &attr, 0.5f);

.....
```

## Remark

None

## Related Items

[njOverhauserSpline](#)  
[njKochanekSpline](#)

**njCeil** Returns the smallest integer not less than n (Ceiling Function).

---

## FORMAT

```
#include <Ninja.h>
Float njCeil( n )
Float n
```

## PARAMETERS

**n**

arbitrary real number

## RETURN VALUE

Smallest integer not less than n

## FUNCTION

Determines the smallest integer that is not less than real number n and returns the value found as the result.

## EXAMPLE

```
y1 = njCeil( 2.8f );
y2 = njCeil( -2.8f );
% Result %
y1 = 3.000000
y2 = -2.000000
```

## NOTES

## RELATED TOPICS

[njFloor](#)

[njRoundOff](#)

# **njCombination**

# Combination function

---

## **Syntax**

```
#include <Ninja.h>
int    njCombination(int n, int k);
int    n
int    k
```

## **Arguments**

**n**

First argument

**k**

Second argument

## **Return Value**

Combination

## **Error**

None

## **Description**

Find out combination of extracting k units from n units ( $nCk = n! / (k!(n-k)!)$ ).

## **Usage Example**

```
#include <NINJA.H>
int n;
.....
n = njCombination(4, 3); /* n = 4 */
.....
```

## **Remark**

None

## **Related Items**

njCombination

## njCos

Returns a cosine.

---

### FORMAT

```
#include <Ninja.h>
Float njCos( ang )
Angle ang
```

### PARAMETERS

**ang**

angle

### RETURN VALUE

value of cosine

### FUNCTION

- Finds the cosine of arbitrary angle ang and returns the value found as the result.

### EXAMPLE

```
ans = njCos( 0x1555 );
% Result %
ans = 0.866041
```

### NOTES

### RELATED TOPICS

[njSin](#)

## **njCosec**

Returns the cosecant.

---

### **FORMAT**

```
#include <Ninja.h>
Float njCosec( ang )
Angle ang
```

### **PARAMETERS**

**ang**

angle

### **RETURN VALUE**

Value of cosecant

### **FUNCTION**

- Determines the cosecant of arbitrary angle ang and returns the value found as the result.

### **EXAMPLE**

```
ans = njCosec( 0x1555 );
% Result %
ans = 2.000112
```

### **NOTES**

### **RELATED TOPICS**

[njCot](#)



## njCosech

Returns the hyperbolic cosecant.

---

### FORMAT

```
#include <Ninja.h>
Float njCosech( n )
Angle n
```

### PARAMETERS

**n**

angle

### RETURN VALUE

Value of hyperbolic cosecant

### FUNCTION

- Determines the hyperbolic cosecant of arbitrary angle n and returns the value found as the result.

### EXAMPLE

```
ans = njCosch( NJM_RAD_ANG(1) );
% Result %
ans = 0.850959
```

### NOTES

### RELATED TOPICS

[njCosh](#)

[njCoth](#)

[njSech](#)

[njSinh](#)

## njCosh

Returns the hyperbolic cosine.

---

### FORMAT

```
#include <Ninja.h>
Float njCosh( n )
Angle n
```

### PARAMETERS

**n**

angle

### RETURN VALUE

Value of hyperbolic cosine

### FUNCTION

- Determines the hyperbolic cosine of given angle n and returns the value found as the result.

### EXAMPLE

```
ans = njCosh( NJM_RAD_ANG(1) );
% Result %
ans = 1.543038
```

### NOTES

### RELATED TOPICS

[njCosech](#)

[njCoth](#)

[njSech](#)

[njSinh](#)

## njCot

Returns the cotangent.

---

### FORMAT

```
#include <Ninja.h>
Float njCot( ang )
Angle ang
```

### PARAMETERS

**ang**

angle

### RETURN VALUE

Value of cotangent

### FUNCTION

- Determines the cotangent of arbitrary angle ang and returns the value found as the result.

### EXAMPLE

```
ans = njCot( NJM_RAD_ANG(1) );
% Result %
ans = 0.642144
```

### NOTES

### RELATED TOPICS

njCosec

## njCoth

Returns the hyperbolic cotangent.

---

### FORMAT

```
#include <Ninja.h>
Float njCoth( n )
Angle n
```

### PARAMETERS

**n**

angle

### RETURN VALUE

Value of the hyperbolic cotangent

### FUNCTION

- Determines the hyperbolic cotangent of arbitrary angle n and returns the value found as the result.

### EXAMPLE

```
ans = njCoth( NJM_RAD_ANG(1) );
% Result %
ans = 1.313062
```

### NOTES

### RELATED TOPICS

[njCosech](#)

[njCosh](#)

[njSech](#)

[njSinh](#)

## njCubicBezier

## 3D Bezier spline correction function

---

### Syntax

```
#include <Ninja.h>
void njCubicBezier(float *idata, float *odata, NJS_SPLINE *attr, float frame);
float      *idata
float      *odata
NJS_SPLINE *attr
float      frame
```

### Arguments

**idata**

Input data (but idata[3\*4])

**odata**

Output data (but odata[3])

**attr**

Spline attribute (not required)

**frame**

Frame

### Return Value

None

### Error

None

### Description

Bezier curve correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.

## Usage Example

```
#include <NINJA.H>
#define X1      1.f
#define Y1      0.f
#define Z1      0.f
#define X2      5.f
.....
float idata[3][4] =
{
    {X1, Y1, Z1},
    {X2, Y2, Z2},
    {X3, Y3, Z3},
    {X4, Y4, Z4},
};
float odata[3];
.....
njCubicBezier((float*)idata, odata, NULL, 0.5f);
.....
```

## Remark

None

## Related Items

[njBezier](#)

## njExp

Returns exponents.

---

### FORMAT

```
#include <Ninja.h>
Float njExp( x )
Float x
```

### PARAMETERS

**x**

an arbitrary real number

### RETURN VALUE

Returns the exponent value of x

### FUNCTION

- Determines the exponent value of the arbitrary real number x and returns the value found as the result.

### EXAMPLE

```
ans = njExp( 2.302585093f );
Result
ans = 10.000000
```

### NOTES

### RELATED TOPICS

## **njFloor**      Returns the largest integer not greater than n (Floor Function).

---

### **FORMAT**

```
#include <Ninja.h>
Float njFloor( n )
Float n
```

### **PARAMETERS**

**n**

arbitrary real number

### **RETURN VALUE**

Largest integer that is not greater than n.

### **FUNCTION**

- Determines the largest integer that is not larger than real number n and returns the value found as the result.

### **EXAMPLE**

```
y1 = njFloor( 2.8f );
y2 = njFloor( -2.8f );
% Result %
y1 = 2.000000
y2 = -3.000000
```

### **NOTES**

### **RELATED TOPICS**

[njCeil](#)

[njRoundOff](#)



## njFraction

Returns the decimal fraction.

---

### FORMAT

```
#include <Ninja.h>
Float njFraction( n )
Float n
```

### PARAMETERS

**n**

arbitrary real number

### RETURN VALUE

Value of decimal fraction

### FUNCTION

- Returns the decimal fraction of arbitrary real number n.

### EXAMPLE

```
ans1 = njFraction( 3.14159265f );
ans2 = njFraction( -3.14159265f );
% Result %
ans1 = 0.14159265
ans2 = -0.14159265
```

### NOTES

### RELATED TOPICS

## njHermite

## Hermite spline correction function

---

### Syntax

```
#include <Ninja.h>
void njHermite(float *idata, float *odata, NJS_SPLINE *attr, float frame);
float          *idata
float          *odata
NJS_SPLINE     *attr
float          frame
```

### Arguments

**idata**

Input data (but idata[3\*4])

**odata**

Output data (but odata[3])

**attr**

Spline attribute (not required)

**frame**

Frame

### Return Value

None

### Error

None

### Description

Spline curve correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.

## Usage Example

```
#include <NINJA.H>
#define X1  1.f
#define Y1  0.f
#define Z1  0.f
#define X2 10.f
#define Y2  0.f
#define Z2  2.f
#define NX1 1.f
#define NY1 0.f
#define NZ1 0.f
#define NX2 0.8f
#define NY2 0.f
#define NZ2 0.6f
.....
float idata[3][4] =
{
  {X1, Y1, Z1},
  {X2, Y2, Z2},
  {NX1, NY1, NZ1},
  {NX2, NY2, NZ2},
};
float odata[3];
.....
njHermite((float*)idata, odata, NULL, 0.5f);
.....
```

## Remark

None

## Related Items

# **njHypot**

Returns length of a hypotenuse.

---

## **FORMAT**

```
#include <Ninja.h>
Float njHypot( x , y )
Float x
Float y
```

## **PARAMETERS**

**x**

an arbitrary real number

**y**

an arbitrary real number

## **RETURN VALUE**

length of a hypotenuse

## **FUNCTION**

- If the value of the length of x and y are given, determines the length of the hypotenus of the right angled triangle and returns the value found as the result.

## **EXAMPLE**

```
ans = njHypot( 3.0f , 4.0f );
Result
ans = 5.0
```

## **NOTES**

## **RELATED TOPICS**

[njSqrt](#)

## njInvertSqrt

Returns the inverse square root.

---

### FORMAT

```
#include <Ninja.h>
Float njInvertSqrt( n )
Float n
```

### PARAMETERS

**n**  
arbitrary real number

### RETURN VALUE

Inverse square root

### FUNCTION

- Determines the inverse square root ( $1/\text{square root}$ ) of arbitrary real number  $n$  and returns the value found as the result.

### EXAMPLE

```
ans = njInvertSqrt( -3.14159265f );
% Result %
ans = -0.564190
```

### NOTES

- If arbitrary real number  $n$  is negative, the negative value of the inverse square root of the absolute value of  $n$  is returned.

### RELATED TOPICS

[njSqrt](#)

## njKochanekSpline

## Kochanek spline correction function

---

### Syntax

```
#include <Ninja.h>
void njKochanekSpline(float *idata, float *odata, NJS_SPLINE *attr, float frame);
float      *idata
float      *odata
NJS_SPLINE *attr
float      frame
```

### Arguments

#### **idata**

Input data (but idata[3\*4])

#### **odata**

Output data (but odata[3])

#### **attr**

Spline attribute

#### **frame**

Frame

### Return Value

None

### Error

None

### Description

Spline curve correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.

For the attributes, please specify tension, bias, and continuity

at the fparam members (attr->fparam[0], attr->fparam[1],attr->fparam[2])

of the spline attribute structure (each ranging from -1.f to 1.f).

## Usage Example

```
#include <NINJA.H>
#define X1  1.f
#define Y1  0.f
#define Z1  0.f
#define X2  5.f
#define TENSION 0.2f
#define BIAS 0.1f
#define CONT -0.1f
.....
float idata[3][4] =
{
  {X1, Y1, Z1},
  {X2, Y2, Z2},
  {X3, Y3, Z3},
  {X4, Y4, Z4},
};
float odata[3];
NJS_SPLINE attr;
.....
attr.fparam[0] = TENSION;
attr.fparam[1] = BIAS;
attr.fparam[2] = CONT;
njKochanekSpline((float*)idata, odata, &attr, 0.5f);
.....
```

## Remark

None

## Related Items

`njOverhauserSpline`  
`njCardinalSpline`

## njLinear

## Line shape correction function

---

### Syntax

```
#include <Ninja.h>

void  njLinear(float *idata, float *odata, NJS_SPLINE *attr, float frame);

float  *idata
float  *odata
NJS_SPLINE *attr
float  frame
```

### Arguments

#### **idata**

Input data (but idata[3\*2])

#### **odata**

Output data (but odata[3])

#### **attr**

Spline attribute (not required)

#### **frame**

Frame

### Return Value

None

### Error

None

### Description

Line shape correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.



## Usage Example

```
#include <NINJA.H>
#define X1  1.f
#define Y1  0.f
#define Z1  0.f
#define X2 10.f
#define Y2  0.f
#define Z2  2.f

.....

float idata[3][2] =
{
  {X1, Y1, Z1},
  {X2, Y2, Z2},
};
float odata[3];

.....

njLinear((float*)idata, odata, NULL, 0.5f);
.....
```

## Remark

None

## Related Items

# **njLog**

Returns the natural logarithm.

---

## **FORMAT**

```
#include <Ninja.h>
Float njLog( n )
Float n
```

## **PARAMETERS**

**n**

arbitrary real number

## **RETURN VALUE**

natural logarithm of n

## **FUNCTION**

- Determines the natural logarithm of arbitrary real number n and returns the value found as the result.

## **EXAMPLE**

```
ans = njLog( 2.f );
% Result %
ans = 0.693147
```

## **NOTES**

## **RELATED TOPICS**

[njLog10](#)

## njLog10

Returns the base 10 logarithm.

---

### FORMAT

```
#include <Ninja.h>
Float njLog10( n )
Float n
```

### PARAMETERS

**n**

arbitrary real number

### RETURN VALUE

Base 10 logarithm of n

### FUNCTION

- Determines the base 10 logarithm of arbitrary real number n and returns the value found as the result.

### EXAMPLE

```
ans = njLog10( 2.f );
% Result %
ans = 0.301030
```

### NOTES

### RELATED TOPICS

[njLog](#)

## njLog2

Returns the base 2 logarithm.

---

### FORMAT

```
#include <Ninja.h>
Float njLog2( n )
Float n
```

### PARAMETERS

**n**

arbitrary real number

### RETURN VALUE

Base 2 logarithm of n

### FUNCTION

- Determines the base 2 logarithm of arbitrary real number n and returns the value found as the result.

### EXAMPLE

```
ans = njLog2( 2.f );
% Result %
ans = 1.000000
```

### NOTES

### RELATED TOPICS

[njLog](#)

## **njOverhauserSpline**      Overhauser spline correction function

---

### **Syntax**

```
#include <Ninja.h>
void  njOverhauserSpline(float *idata, float *odata, NJS_SPLINE *attr, float frame);
float      *idata
float      *odata
NJS_SPLINE *attr
float      frame
```

### **Arguments**

**idata**

Input data (but idata[3\*4])

**odata**

Output data (but odata[3])

**attr**

Spline attribute (not required)

**frame**

Frame

### **Return Value**

None

### **Error**

None

### **Description**

Spline curve correction value is stored in the specified argument.

Please set the frame number to a value between 0 and 1.

### Usage Example

```
#include <NINJA.H>
#define X1  1.f
#define Y1  0.f
#define Z1  0.f
#define X2  5.f
    .....

float idata[3][4] =
{
  {X1, Y1, Z1},
  {X2, Y2, Z2},
  {X3, Y3, Z3},
  {X4, Y4, Z4},
};
float odata[3];
    .....

njOverhauserSpline((float*)idata, odata, NULL, 0.5f);
    .....
```

### Remark

None

### Related Items

[njCardinalSpline](#)  
[njKochanekSpline](#)

## njPow

Returns the power of a number.

---

### FORMAT

```
#include <Ninja.h>
Float njPow( n1, n2 )
Float n1
Float n2
```

### PARAMETERS

**n1**

arbitrary real number 1

**n2**

arbitrary real number 2

### RETURN VALUE

value of n1 to the n2nd power

### FUNCTION

- Calculates the value of arbitrary real number n1 raised to the power of arbitrary real number n2 and returns the value found as the result.

### EXAMPLE

```
void main( void )
{
Float x = 2.f , y = 4.f
Float ans;

ans = njPow( x , y );
printf( "ans = %f\n", ans );
}
ans = 16.000000
```

### NOTES

### RELATED TOPICS

# **njRandom**

Generates a random number.

---

## **FORMAT**

```
#include <Ninja.h>
Float njRandom( )
```

## **PARAMETARS**

random number

## **RETURN VALUE**

Real number in the range  $0 \leq X < 1$

## **FUNCTION**

- Generates a random number in the range  $0 \leq X < 1$  and returns the generated value as the result.

## **EXAMPLE**

```
Float ret;
.....

njRandomSeed( (unsigned)time( NULL ) );
ret = njRandom();
```

## **NOTES**

## **RELATED TOPICS**

[njRandomSeed](#)



## **njRandomSeed**

Sets the random number seed.

---

### **FORMAT**

```
#include <Ninja.h>
void njRandomSeed( n )
Uint32 n
```

### **PARAMETERS**

**n**

random number seed

### **RETURN VALUE**

None

### **FUNCTION**

- Sets the specified arbitrary integer n as the random number seed.

### **EXAMPLE**

```
Float ret;
.....

njRandomSeed( (unsigned)time( NULL ) );
ret = njRandom();
```

### **NOTES**

### **RELATED TOPICS**

# **njRoundOff**

Rounds down the decimal fraction.

---

## **FORMAT**

```
#include <Ninja.h>
Float njRoundOff( n )
Float n
```

## **PARAMETERS**

**n**

arbitrary real number

## **RETURN VALUE**

Integer resulting from rounding down

## **FUNCTION**

- Truncates the decimal portion of arbitrary real number n and returns the integer portion as the result.

## **EXAMPLE**

```
y1 = njRoundOff( 2.8f );
y2 = njRoundOff( -2.8f );
% Result %
y1 = 2.000000
y2 = -2.000000
```

## **NOTES**

## **RELATED TOPICS**

[njCeil](#)

[njFloor](#)

[njRoundUp](#)

## njRoundUp

Rounds up the decimal fraction.

---

### FORMAT

```
#include <Ninja.h>
Float njRoundUp( n )
Float n
```

### PARAMETERS

**n**

Arbitrary real number

### RETURN VALUE

Integer resulting from rounding up

### FUNCTION

- Rounds up the decimal fraction of arbitrary real number n and returns the resulting integer.

### EXAMPLE

```
y1 = njRoundUp( 2.8f );
y2 = njRoundUp( -2.8f );
% Result %
y1 = 3.000000
y2 = -3.000000
```

### NOTES

### RELATED TOPICS

[njCeil](#)

[njFloor](#)

[njRoundOff](#)

# **njSec**

Returns a secant.

---

## **FORMAT**

```
#include <Ninja.h>
Float njSec( ang )
Angle ang
```

## **PARAMETERS**

**ang**

angle

## **RETURN VALUE**

Value of secant

## **FUNCTION**

- Determines the secant of arbitrary angle n and returns the value found as the result.

## **EXAMPLE**

```
ans = njSec( NJM_RAD_ANG(1) );
% Result %
ans = 1.850710
```

## **NOTES**

## **RELATED TOPICS**

njCosec

## njSech

Returns a hyperbolic secant.

---

### FORMAT

```
#include <Ninja.h>
Float njSech( ang )
Angle ang
```

### PARAMETERS

**ang**

angle

### RETURN VALUE

Value of hyperbolic secant

### FUNCTION

- Determines the hyperbolic secant of arbitrary angle ang and returns the value found as the result.

### EXAMPLE

```
ans = njSech( NJM_RAD_ANG(1) );
% Result %
ans = 0.648072
```

### NOTES

### RELATED TOPICS

[njCosech](#)

[njCosh](#)

[njCoth](#)

[njSinh](#)

[njTanh](#)

# njSin

Returns a sine.

---

## FORMAT

```
#include <Ninja.h>
Float njSin( ang )
Angle ang
```

## PARAMETERS

**ang**

angle

## RETURN VALUE

Value of sine

## FUNCTION

- Determines the sine of arbitrary angle ang and returns the value found as the result.

## EXAMPLE

```
ans = njSin( 0x1555 );
% Result %
ans  = 0.499972
```

## NOTES

## RELATED TOPICS

[njCos](#)

[njTan](#)

## njSinh

Returns a hyperbolic sine.

---

### FORMAT

```
#include <Ninja.h>
Float njSinh( ang )
Angle ang
```

### PARAMETERS

**ang**

angle

### RETURN VALUE

Value of hyperbolic sine

### FUNCTION

- Determines the hyperbolic sine of given angle ang and returns the value found as the result.

### EXAMPLE

```
ans = njSinh( NJM_RAD_ANG(1) );
% Result %
ans = 1.175145
```

### NOTES

### RELATED TOPICS

[njCosech](#)

[njCosh](#)

[njCoth](#)

[njSech](#)

[njTanh](#)

# njSqrt

Returns a square root.

---

## FORMAT

```
#include <Ninja.h>
Float njSqrt( n )
Float n
```

## PARAMETERS

**n**

arbitrary real number

## RETURN VALUE

Value of square root

## FUNCTION

- Determines the square root of a given real number and returns the value found as the result.

## EXAMPLE

```
ans = njSqrt( -3.14159265f );
% Result %
ans = -1.772454
```

## NOTES

- If the value of arbitrary real number n is negative, the negative of the square root of the absolute value of n is returned.

## RELATED TOPICS

[njHypot](#)



# njTan

Returns a tangent.

---

## FORMAT

```
#include <Ninja.h>
Float njTan( ang )
Angle ang
```

## PARAMETERS

**ang**

angle

## RETURN VALUE

Value of tangent

## FUNCTION

- Determines the tangent of arbitrary angle ang and returns the value found as the result.

## EXAMPLE

```
ans = njTan( 0x1555 );
% Result %
ans = 0.577307
```

## NOTES

## RELATED TOPICS

njCos

njSin

## njTanh

Returns a hyperbolic tangent.

---

### FORMAT

```
#include <Ninja.h>
Float njTanh( ang )
Angle ang
```

### PARAMETERS

**ang**

angle

### RETURN VALUE

Value of hyperbolic tangent

### FUNCTION

- Determines the hyperbolic tangent of given angle ang and returns the value found as the result.

### EXAMPLE

```
ans = njTanh( NJM_RAD_ANG(1) );
% Result %
ans = 0.761579
```

### NOTES

### RELATED TOPICS

[njCosech](#)

[njCosh](#)

[njCoth](#)

[njSech](#)

[njSinh](#)



## *5. 2D Graphics Functions*

---

### **Contents**

njDrawCircle2D .....	Draws circles on a 2D screen. ....	NLS-168
njDrawLine2D .....	Draws lines on a 2D screen. ....	NLS-171
njDrawPoint2D.....	Draws points on a 2D screen. ....	NLS-173
njDrawPolygon2D .....	Draws a polygon on a 2D screen. ....	NLS-175
njDrawTriangle2D.....	Draws triangles on a 2D screen. ....	NLS-177

## njDrawCircle2D

Draws circles on a 2D screen.

---

### FORMAT

```
#include <Ninja.h>
void njDrawCircle2D( *p, n, pri, attr )
NJS_POINT2COL *p
Int n
Float pri
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of circles to be drawn

**n**

number of circles to be drawn

**pri**

priority (used as the Z value)

**attr**

attributes (specifies drawing method)

### RETURN VALUE

None

## FUNCTION

- Draws circles.
- The following attributes can be used.

<b>NJD_DRAW_NORMAL</b>	An ellipse is drawn with center at x,y coordinates of the 1st point. The x of the 2nd point defines the distance from the center in the x direction, and the y defines the distance from the center in the y direction.
<b>NJD_DRAW_INSCRIBED</b>	The circle is inscribed in the rectangle whose diagonal is defined by two points.
<b>NJD_DRAW_FAN</b>	Allows drawing of concentric circles.
<b>NJ_DRAW_CONNECTED</b>	Draw continuously.
<b>NJD_FILL</b>	Drawing with fill.
<b>NJD_TRANSPARENT</b>	Transparent drawing.
<b>NJD_USE_TEXTURE</b>	Drawing with texture.

### Example)

- NJD\_DRAW\_INSCRIBED | NJD\_DRAW\_FAN

Allows drawing of circles radially around a center defined by the 1st point.

- NJD\_DRAW\_INSCRIBED | NJD\_DRAW\_CONNECTED

Allows drawing of consecutive circles inscribed in rectangles.

## EXAMPLE

The following draws four inscribed circles centered on a point.

```
Sint32 i;
NJS_POINT2COL    p2c[5];
NJS_POINT2       p[5];
NJS_COLOR        col[5];

p2c.p = p;
p2c.col = col;
p2c.tex = NULL;
p2c.num = 5;

p2c.p[0].x = 320;
p2c.p[0].y = 240;
p2c.col[0].color = 0xff000077;

p2c.p[1].x = 70;
p2c.p[1].y = 40;
p2c.col[1].color = 0xff000077;
```

```
p2c.p[2].x = 570;  
p2c.p[2].y = 40;  
p2c.col[2].color = 0x00ff0077;
```

```
p2c.p[3].x = 570;  
p2c.p[3].y = 440;  
p2c.col[3].color = 0x0000ff77;
```

```
p2c.p[4].x = 70;  
p2c.p[4].y = 440;  
p2c.col[4].color = 0xffffffff77;
```

```
njDrawCircle2D(p,4,-10.f,NJD_DRAW_NORMAL|NJD_DRAW_INSCRIBED|NJD_DRAW_FAN);
```

## NOTES

- The NJD\_DRAW\_NORMAL may be omitted when used in combination with other attributes.

## RELATED TOPICS

NJS\_POINT2COL

## njDrawLine2D

Draws lines on a 2D screen.

---

### FORMAT

```
#include <Ninja.h>
void njDrawLine2D( *p, n, pri, attr )
NJS_POINT2COL *p
Int n
Float pri
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of end points of lines to be drawn

**n**

number of lines to be drawn

**pri**

priority (used as Z value)

**attr**

attributes (specifies drawing method)

### RETURN VALUE

None

### FUNCTION

- Draws n lines.
- Valid attributes are used to draw n lines using point data given in the screen coordinate system.

NJD_DRAW_NORMAL	Drawing of independent lines.
NJD_DRAW_FAN	Drawing of radial lines.
NJD_DRAW_CONNECTED	Drawing of connected lines.
NJD_TRANSPARENT	Translucent drawing.
NJD_USE_TEXTURE	Drawing with texture.

The following five attributes can be specified. These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

### EXAMPLE

```
Sint32 i;
NJS_POINT2COL    p2c[100];
NJS_POINT2       p[100];
NJS_COLOR        col[100];

p2c.p = p;
p2c.col = col;
p2c.tex = NULL;
p2c.num = 100;

for(i=0;i<100;i++){
    p2c.p[i].x = (Sint16)(njRandom() * 640.f);
    p2c.p[i].y = (Sint16)(njRandom() * 480.f);
    p2c.p[i].col.color = (Sint32)(njRandom() * 0xFFFFFFFF);
}

njDrawLine2D(p,100,-2.f,NJD_DRAW_NORMAL|NJD_TRANSPARENT);
```

### NOTES

- The NJD\_DRAW\_NORMAL attribute can be omitted when it is used in combination with other attributes.

### RELATED TOPICS

NJS\_POINT2COL



## **njDrawPoint2D**

Draws points on a 2D screen.

---

### **FORMAT**

```
#include <Ninja.h>
void njDrawPoint2D( *p, n, pri, attr)
NJS_POINT2COL *p
Int n
Float pri
Uint32 attr
```

### **PARAMETERS**

**\*p**

list of coordinates of points to be drawn

**n**

number of points to draw

**pri**

priority (used as Z value)

**attr**

attributes (specifies drawing method)

### **RETURN VALUE**

None

## FUNCTION

- Draws n points.
- An array of NJS\_POINT2COL type is used when drawing multiple points.
- Valid attributes are used to draw n points of the specified color in the screen coordinate system at the coordinates specified by p.
- The following three attributes can be specified.

NJD_DRAW_NORMAL	Normal drawing.
NJD_TRANSPARENT	Transparent drawing.
NJD_USE_TEXTURE	Drawing with texture.

These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

## EXAMPLE

The following draws 100 points at random locations.

```
Sint32 i;
NJS_POINT2COL  p2c[100];
NJS_POINT2     p[100];
NJS_COLOR      col[100];

p2c.p = p;
p2c.col = col;
p2c.tex = NULL;
p2c.num = 100;

for(i=0;i<100;i++){
    p2c.p[i].x = (Sint16)(njRandom() * 640.f);
    p2c.p[i].y = (Sint16)(njRandom() * 480.f);
    p2c.col[i].color = (Sint32)(njRandom() * 0xFFFFFFFF);
}

njDrawPoint2D(p,100,-2.f,NJD_DRAW_NORMAL);
```

## NOTES

- The NJD\_DRAW\_NORMAL attribute may be omitted when used in combination with other attributes.

## RELATED TOPICS

## njDrawPolygon2D

Draws a polygon on a 2D screen.

---

### FORMAT

```
#include <Ninja.h>
void njDrawPolygon2D( *p, n, pri, attr )
NJS_POINT2COL *p
Int n
Float pri
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of vertices of polygons to be drawn

**n**

number of vertices of polygons

**pri**

priority (used as Z value)

**attr**

attributes (specifies the drawing method)

### RETURN VALUE

None

### FUNCTION

- Draws a polygon of n sides.
- The following four attributes can be used.

NJD_DRAW_NORMAL	Normal drawing.
NJD_FILL	Drawing with interior fill.
NJD_TRANSPARENT	Transparent drawing.
NJD_USE_TEXTURE	Drawing with texture.

These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

### EXAMPLE

The following draws a hexagon.

```
Sint32 i;  
NJS_POINT2COL    p2c[6];  
NJS_POINT2       p[6];  
NJS_COLOR        col[6];  
  
p2c.p = p;  
p2c.col = col;  
p2c.tex = NULL;  
p2c.num = 6;  
  
p2c.p[0].x = 427;  
p2c.p[0].y = 116;  
p2c.col[0].color = 0xFF000055;  
  
p2c.p[1].x = 555;  
p2c.p[1].y = 200;  
p2c.col[1].color = 0xFFFFF55;  
  
p2c.p[2].x = 555;  
p2c.p[2].y = 285;  
p2c.col[2].color = 0x00FF0055;  
  
p2c.p[3].x = 427;  
p2c.p[3].y = 371;  
p2c.col[3].color = 0xFFFFF55;  
  
p2c.p[4].x = 300;  
p2c.p[4].y = 285;  
p2c.col[4].color = 0x0000FF55;  
  
p2c.p[5].x = 300;  
p2c.p[5].y = 200;  
p2c.col[5].color = 0xFFFFF55;  
  
njDrawPolygon2D(p,6,-2.f,NJD_DRAW_NORMAL|NJD_FILL);
```

### NOTES

The NJD\_DRAW\_NORMAL attribute may be omitted when it is used in combination with other attributes.

### RELATED TOPICS

## njDrawTriangle2D

Draws triangles on a 2D screen.

---

### FORMAT

```
#include <Ninja.h>
void njDrawTriangle2D( *p, n, pri, attr )
NJS_POINT2COL *p
Int n
Float pri
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of points to be drawn

**n**

number of points to draw

**pri**

priority (used as Z value)

**attr**

attributes (specifies drawing method)

### RETURN VALUE

None

### FUNCTION

- Draws n triangles.
- The following six attributes can be used.

NJD_DRAW_NORMAL	Draws separate triangles.
NJD_DRAW_FAN	Draws triangles radially.
NJD_DRAW_CONNECTED	Draws connected triangles.
NJD_FILL	Drawing with filled interior.
NJD_TRANSPARENT	Translucent drawing.

### NJD\_USE\_TEXTURE

Drawing with texture.

- These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

## EXAMPLE

The following draws four connected triangles.

```
Sint32 i;
NJS_POINT2COL    p2c[6];
NJS_POINT2       p[6];
NJS_COLOR        col[6];

p2c.p = p;
p2c.col = col;
p2c.tex = NULL;
p2c.num = 6;

p2c.p[0].x = 100;
p2c.p[0].y = 300;
p2c.col[0].color = 0xFFFF00AA;

p2c.p[1].x = 150;
p2c.p[1].y = 100;
p2c.col[1].color = 0x0FFFF000;

p2c.p[2].x = 200;
p2c.p[2].y = 300;
p2c.col[2].color = 0x00FFFF00;

p2c.p[3].x = 250;
p2c.p[3].y = 100;
p2c.col[3].color = 0xF00FFF55;

p2c.p[4].x = 300;
p2c.p[4].y = 300;
p2c.col[4].color = 0xFF00FF00;

p2c.p[5].x = 350;
p2c.p[5].y = 100;
p2c.col[5].color = 0xFFFF00F00;

njDrawTriangle2D(p,4,-3.f,NJD_DRAW_NORMAL|NJD_DRAW_CONNECTED|NJD_FILL);
```

## NOTES

## RELATED TOPICS



## *6. 3D Graphics Functions*

---

### **Contents**

njDrawLine3D .....	Draws lines in 3D space. ....	NLS-180
njDrawPoint3D.....	Draws points in 3D space. ....	NLS-182
njDrawPolygon3D .....	Draws a polygon in 3D space. ....	NLS-184
njDrawTriangle3D.....	Draws triangles in 3D space. ....	NLS-186

# njDrawLine3D

Draws lines in 3D space.

## FORMAT

```
#include <Ninja.h>
void njDrawLine3D(*p, n, attr)
NJS_POINT3COL *p
Int n
Uint32 attr
```

## PARAMETERS

**\*p**

list of coordinates of endpoints of lines to be drawn

**n**

number of lines to be drawn

**attr**

attributes (specifies the drawing method)

## RETURN VALUE

None

## FUNCTION

- Draws n lines.
- The following 5 attributes can be used for drawing.

NJD_DRAW_NORMAL	Normal drawing.
NJD_DRAW_CONNECTED	Drawing of connected lines.
NJD_DRAW_WHEEL	Radial drawing.
NJD_TRANSPARENT	Translucent drawing.
NJD_USE_TEXTURE	Drawing with texture.

These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.



## EXAMPLE

The following draws 50 lines at random.

```
int i;
NJS_POINT3COL p;
NJS_POINT3 point[100];
NIS_COLOR color[100];

p->p = point;
p->col = color;
p->tex = NULL;
p.num = 100;

for(i = 0; i < 100; i++){
    p->p[i].x = njRandom()*1000.f-500.f;
    p->p[i].y = njRandom()*1000.f-500.f;
    p->p[i].z = -njRandom()*1000.f;
    p->col[i].argb.a = (Uint8)(0x80*njRandom());
    p->col[i].argb.r = (Uint8)(0x80*njRandom());
    p->col[i].argb.g = (Uint8)(0x80*njRandom());
    p->col[i].argb.b = (Uint8)(0x80*njRandom());
}

njDrawLine3D(&p, 100, NJD_DRAW_NORMAL|NJD_TRANSPARENT);
```



**NOTE:** Since this function draws in 3D, the view, screen, and matrix stack settings must be made before using it.

---

## NOTES

The NJD\_DRAW\_NORMAL attribute can be omitted when used in combination with other attributes.

## RELATED TOPICS

## njDrawPoint3D

Draws points in 3D space.

---

### FORMAT

```
#include <Ninja.h>
void njDrawPoint3D(*p, n, attr)
NJS_POINT3COL *p
Int n
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of points to be drawn.

**n**

number of points to be drawn

**attr**

attributes (specifies the drawing method)

### RETURN VALUE

None

### FUNCTION

- Draws *n* points.
- An array of NJS\_POINT2COL type is used when drawing multiple points. *n* points of the specified color are drawn at the points specified by *p* in the global coordinate system.

NJD_DRAW_NORMAL	Normal drawing.
NJD_TRANSPARENT	Transparent drawing.
NJD_USE_TEXTURE	Drawing with texture.

The following 3 attributes can be used for drawing. These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

## EXAMPLE

The following draws 100 points at random.


```
int i;
NJS_POINT3COL p;
NJS_POINT3 point[100];
NIS_COLOR color[100];

p->p = point;
p->col = color;
p->tex = NULL;
p.num = 100;

for(i = 0; i < 100; i++){
    p -> p[i].x = njRandom()*1000.f-500.f;
    p -> p[i].y = njRandom()*1000.f-500.f;
    p -> p[i].z = -njRandom()*1000.f;
    p -> col[i].argb.a = (Uint8)(0x80*njRandom());
    p -> col[i].argb.r = (Uint8)(0x80*njRandom());
    p -> col[i].argb.g = (Uint8)(0x80*njRandom());
    p -> col[i].argb.b = (Uint8)(0x80*njRandom());
}

njDrawPoint3D(&p, 100, NJD_DRAW_NORMAL|NJD_TRANSPARENT);
```

---

 **NOTE:** Since this function draws in 3D, the view, screen, and matrix stack settings must be made before using it.

---

## NOTES

The NJD\_DRAW\_NORMAL may be omitted when it is used in combination with other attributes.

## RELATED TOPICS

## njDrawPolygon3D

Draws a polygon in 3D space.

---

### FORMAT

```
#include <Ninja.h>
void njDrawPolygon3D(*p, n, attr)
NJS_POINT3COL *p
Int n
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of vertices of a polygon to be drawn

**n**

number of vertices of a polygon

**attr**

attributes (specifies the drawing method)

### RETURN VALUE

None

### FUNCTION

- Draws a polygon of n sides.
- The following four attributes can be used.

NJD_DRAW_NORMAL	Normal drawing.
NJD_FILL	Drawing with interior fill.
NJD_TRANSPARENT	Transparent drawing.
NJD_USE_TEXTURE	Drawing with texture.

These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

## EXAMPLE

The following draws 100 squares at random.

```
int i;

NJS_POINT3COL p;
NJS_POINT3 point[4];
NJS_COLOR color[4];

p->p = point;
p->col = color;
p->tex = NULL;
p.num = 4;

for(i = 0; i < 100; i++){
    int j;
    p->p[0].x = njRandom()*1000.f-500.f;
    p->p[0].y = njRandom()*1000.f-500.f;
    p->p[0].z = -3000.f+(float)i;
    p->p[1].x = p->p[0].x-polysize;
    p->p[1].y = p->p[0].y;
    p->p[1].z = p->p[0].z;
    p->p[2].x = p->p[0].x-polysize;
    p->p[2].y = p->p[0].y-polysize;
    p->p[2].z = p->p[0].z;
    p->p[3].x = p->p[0].x;
    p->p[3].y = p->p[0].y-polysize;
    p->p[3].z = p->p[0].z;

    for(j = 0; j < 4; j++){
        p->col[j].argb.a =
        (UInt8)(0x80*njRandom());
        p->col[j].argb.r =
        (UInt8)(0x80*njRandom());
        p->col[j].argb.g =
        (UInt8)(0x80*njRandom());
        p->col[j].argb.b =
        (UInt8)(0x80*njRandom());
    }
    njDrawPolygon3D(&p, 4, NJD_DRAW_NORMAL|NJD_FILL);
}
```



**NOTE:** Since this function draws in 3D, the view, screen, and matrix stack settings must be made before using it.

---

## NOTES

The `NJD_DRAW_NORMAL` may be omitted when it is used in combination with other attributes.

## RELATED TOPICS

## njDrawTriangle3D

Draws triangles in 3D space.

---

### FORMAT

```
#include <Ninja.h>
void njDrawTriangle3D(*p, n, attr)
NJS_POINT3COL *p
Int n
Uint32 attr
```

### PARAMETERS

**\*p**

list of coordinates of vertices of a polygon to be drawn

**n**

number of triangles to be drawn

**attr**

attributes (specifies the drawing method)

### RETURN VALUE

None

### FUNCTION

- Draws a polygon of n sides.
- The following six attributes can be used.

NJD_DRAW_NORMAL	Normal drawing.
NJD_DRAW_CONNECTED	Draws connected triangles.
NJD_DRAW_FAN	Draws triangles radially.
NJD_FILL	Drawing with filled interior.
NJD_TRANSPARENT	Transparent drawing.
NJD_USE_TEXTURE	Drawing with texture.

These attributes can be used in mutually consistent combinations by using the "|" character as a delimiter.

## EXAMPLE

The following draws 100 triangles at random.

```

#define TRI_NUM 100
#define trisize 50.f

int i;
NJS_POINT3COL p;
NJS_POINT3 point[TRI_NUM*3];
NJS_COLOR color[TRI_NUM*3];
float trisize_half = trisize/2.f;
float triheight;

p->p = point;
p->col = color;
p->tex = NULL;
p.num = TRI_NUM*3;

triheight = trisize_half*njSqrt(3.f);
i = 0;
do{
    p->p[i].x = njRandom()*1000.f-500.f;
    p->p[i].y = njRandom()*1000.f-500.f;
    p->p[i].z = -3000.f+(float)i;
    p->col[i].argb.a = (Uint8)(0x80*njRandom());
    p->col[i].argb.r = (Uint8)(0x80*njRandom());
    p->col[i].argb.g = (Uint8)(0x80*njRandom());
    p->col[i+1].argb.b = (Uint8)(0x80*njRandom());
    p->p[i].x = p[i-1].p.x+trisize_half;
    p->p[i].y = p[i-1].p.y+triheight;
    p->p[i].z = p[i-1].p.z;
    p->col[i].argb.a = (Uint8)(0x80*njRandom());
    p->col[i].argb.r = (Uint8)(0x80*njRandom());
    p->col[i].argb.g = (Uint8)(0x80*njRandom());
    p->col[i+1].argb.b = (Uint8)(0x80*njRandom());
    p->p[i].x = p[i-2].p.x-trisize_half;
    p->p[i].y = p[i-2].p.y+triheight;
    p->p[i].z = p[i-2].p.z;
    p->col[i].argb.a = (Uint8)(0x80*njRandom());
    p->col[i].argb.r = (Uint8)(0x80*njRandom());
    p->col[i].argb.g = (Uint8)(0x80*njRandom());
    p->col[i+1].argb.b = (Uint8)(0x80*njRandom());
while(i < TRI_NUM*3);
njDrawTriangle3D(&p,TRI_NUM, NJD_DRAW_NORMAL|NJD_FILL);

```



**Note:** Since this function draws in 3D, the view, screen, and matrix stack settings must be made before using it.

## NOTEMS

The NJD\_DRAW\_NORMAL may be omitted when it is used in combination with other attributes.

## RELATED TOPICS







## 7. Light Functions

---

### Contents

<b>njSetLightAlpha</b> .....	Sets the changes of alpha against material, whose light is set by njCreateLight. ....	NLS-190
<b>njCreateLight</b> .....	Defines a light source type and registers a new light. ....	NLS-192
<b>njDeleteAllLight</b> .....	<b>This function deletes all the lights set at njCreateLight.</b> ....	NLS-196
<b>njDeleteLight</b> .....	Deletes a light created by njCreateLight. ....	NLS-198
<b>njLightAllOn</b> .....	<b>This function reflects the light set at njCreateLight (i.e. it turns on the lights).</b> ....	NLS-199
<b>njLightAllOff</b> .....	<b>This function does not reflect all the lights set at njCreateLight (i.e. it turns off the lights).</b> ....	NLS-201
<b>njLightOff</b> .....	Deactivates a light created by njCreateLight (turns the light off). ....	NLS-203
<b>njLightOn</b> .....	Activates a light created by njCreateLight (turns the light on). ....	NLS-204
<b>njMultiLightMatrix</b> .....	Multiplies a matrix with a light matrix. ....	NLS-206
<b>njRotateLightX</b> .....	Rotates a light matrix around the X axis. ....	NLS-208
<b>njRotateLightXYZ</b> .....	Rotates a light matrix around the X, Y, and Z axes. ....	NLS-210
<b>njRotateLightY</b> .....	Rotates a light matrix around the Y axis. ....	NLS-212
<b>njRotateLightZ</b> .....	Rotates a light matrix around the Z axis. ....	NLS-214
<b>njSetLightAngle</b> .....	Sets the limit angle of a light created with njCreateLight. ....	NLS-216
<b>njSetLightColor</b> .....	Sets the color of light defined by njCreateLight. ....	NLS-218
<b>njSetLightDirection</b> .....	Sets the direction of light defined by njCreateLight. ....	NLS-220
<b>njSetLightIntensity</b> .....	Sets the intensity of light defined using njCreateLight. ....	NLS-222
<b>njSetLightLocation</b> .....	Sets the location of light defined by njCreateLight. ....	NLS-224
<b>njSetLightRange</b> .....	Sets the limit distance of a light created with njCreateLight. ....	NLS-226
<b>njSetUserLight</b> .....	Assigns a user-defined light function to a light. ....	NLS-228
<b>njTranslateLight</b> .....	Applies a matrix that gives parallel translation along each axis. ....	NLS-233
<b>njTranslateLightV</b> .....	Moves a light matrix laterally according to a directional vector. ....	NLS-235
<b>njUnitLightMatrix</b> .....	Unitizes a light matrix. ....	NLS-237

## **njSetLightAlpha**

Sets the changes of alpha against material, whose light is set by njCreateLight.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetLightAlpha( *ptr, rate )
NJS_LIGHT *ptr
Float rate
```

### **PARAMETERS**

**\*ptr**

light pointer

**rate**

magnification

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets the changes of transparency (alpha) produced by specular.
- Shows the change that as the light specular luminance increases, the material alpha value decreases (more opaque).
- Argument default value, which is set by CreateLight, is 0 (no reflection). The argument should be set with magnification against the specular luminance (specular luminance x magnification).
- The result will be added to the material alpha value.

## EXAMPLE

```
#include
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_DIR_LIGHT);
.....
/* Decreases the alpha by 0.5 of specular luminance. */
njSetLightAlpha(&light, 0.5f);
```

## NOTES

- Light must be specified by njCreateLight.
- If it is specified by either njFastDrawModel, njFastDrawObject, njFastDrawMotion, or njFastAction, this function has no effect.

## RELATED TOPICS

`njCreateLight()`

## **njCreateLight**      Defines a light source type and registers a new light.

---

### **FORMAT**

```
#include <NINJA.H>
void njCreateLight( *ptr, lsrc )
NJS_LIGHT *ptr
Int lsrc
```

### **PARAMETERS**

**\*ptr**

light pointer

**lsrc**

light type

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- The principle factors involved in light calculation are the directions of rays and lines normal to polygons (or to vertices), and the distance between the light and the vertices (or representative point in the model). In principle, the angle between the incident rays and the lines normal to an illuminated polygon will be 90 to 180 degrees. The closer the angle is to 180 degrees, the brighter the lighting, and illumination completely vanishes if the angle becomes less than 90 degrees. Further illumination is attenuated with distance (except with parallel light sources).



**NOTE:** The angular calculation range for NJD\_DIR\_LIGHT|NJD\_SIMPLE\_LIGHT only is 0 to 180 degrees.

---

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();

/* Define parallel light source */
njCreateLight(&light, NJD_DIR_LIGHT);
.....

/* A parallel light source remains effective
 * for all model functions until njDeleteLight
 * or AnjCreateLightOff is called.
 */
njDrawObject(...);
.....
```

## NOTES

Currently, `njDrawObject` and `njDrawModel` are required in order for lighting to be effective. The light types (lsrc) are as follows.

### 1. Single light model

#### **NJD\_AMBIENT**

Ambient light: All surfaces are equally illuminated.

#### **NJD\_DIR\_LIGHT**

Parallel light: Illumination is unaffected by distance in the direction specified by `njSetLightDirection` or in the direction resulting from matrix rotation by functions such as `njRotateLightX`.

The light flag `NJD_SIMPLE_LIGHT` can be added to this type of light source.

#### **NJD\_POINT\_LIGHT**

Point light source: Radial illumination from the point source specified by `njSetLightLocation` or from the translated point location resulting from matrix transformations such as `njTranslateLight`.

The light flags `NJD_SIMPLE_LIGHT` and `NJD_BLOCK_LIGHT` can be added to this type of light source.

### **NJD\_SPOT\_LIGHT**

Spotlight: With this light source type, the direction and position of lighting are determined in the same manner as with parallel lighting and point lighting.

This type of light source illuminates a circular area whose angular size is determined by the `njSetLightAngle` function. The light flags `NJD_SIMPLE_LIGHT` and `NJD_BLOCK_LIGHT` can be added to this type of light source.

## **2. Compound light models**

### **NJD\_SPEC\_DIR(NJD\_SPEC\_POINT)**

Parallel light source with highlighting (point light source): The angle of specular highlighting is determined by `njSetLightAngle`.

Otherwise, behavior is the same as with parallel lighting (point light source).

The light flag `NJD_SIMPLE_LIGHT` can be added to this type of light source.

(`NJD_BLOCK_LIGHT` can also be specified, but only with point lighting.)

### **NJD\_LAMBERT\_DIR(NJD\_LAMBERT\_POINT)**

Lambert model: This illumination uses a parallel light source (point light source) that is governed by a standard lighting model including ambient light.

The light flag `NJD_SIMPLE_LIGHT` can be added to this type of light source.

(`NJD_BLOCK_LIGHT` can also be specified, but only with point lighting.)

### **NJD\_PHONG\_DIR(NJD\_PHONG\_POINT)**

Phong model: This illumination uses a parallel light source (point light source) that is governed by a standard lighting model including ambient light and specular light.

The light flag `NJD_SIMPLE_LIGHT` can be added to this type of light source.

(`NJD_BLOCK_LIGHT` can also be specified, but only with point lighting.)

## **3. Other flags**

### **NJD\_SIMPLE\_LIGHT**

This flag generally makes calculation simpler and faster, but may have an effect on quality.

This flag is specified using an OR operator; for example,

`NJD_SPEC_POINT | NJD_SIMPLE_LIGHT` can be used.

### **NJD\_BLOCK\_LIGHT**

This flag makes light calculation faster limiting distance calculation to a single representative point, rather than to all vertices.

However, it can greatly affect quality, and the expected benefit is not achieved with some models.

This flag is used in the same manner as NJD\_SIMPLE\_LIGHT.

## **4. User-defined (callback) light**

### **NJD\_USER\_LIGHT**

Calls a user-defined light function.

## **RELATED TOPICS**

`njDrawObject`

`njDrawModel()`

## **njDeleteAllLight**

This function deletes all the lights set at njCreateLight.

---

### **Syntax**

```
#include <Ninja.h>
void      njDeleteAllLight( void )
```

### **Parameters**

None

### **Return Value**

None

### **Error**

None

### **Description**

This function deletes all the lights that were registered at the light source system.

However, after the lights are deleted, they can be re-installed again by calling njCreateLight again.



## Example

```
#include <NINJA.H>
.....
NJS_LIGHT light[3];
njInitSystem();
/* Register the three light sources */
njCreateLight(&light[0], NJD_DIR_LIGHT);
njCreateLight(&light[1], NJD_POINT_LIGHT);
njCreateLight(&light[2], NJD_SPOT_LIGHT);
.....
/* Delete all light sources */
njDeleteAllLight();
.....
```

## Remark

None

## Related Items

[njCreateLight](#)  
[njDeleteLight](#)

## **njDeleteLight**

Deletes a light created by njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njDeleteLight( *ptr )
NJS_LIGHT *ptr
```

### **PARAMETERS**

**\*ptr**

light pointer

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Deletes a light defined in the light system.
- However, after deletion, the light source can be restored by recalling njCreateLight.

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;
njInitSystem();

/* Define parallel light source */
njCreateLight(&light, NJD_DIR_LIGHT);
.....

/* Delete parallel light source */
njDeleteLight(&light);
.....
```

### **NOTES**

### **RELATED TOPICS**

[njCreateLight\(\)](#)

## **njLightAllOn**

This function reflects the light set at njCreateLight (i.e. it turns on the lights).

---

### **Syntax**

```
#include <Ninja.h>
void    njLightAllOn( void )
```

### **Parameters**

None

### **Return Value**

None

### **Error**

None

### **Description**

This function turns on all the lights that were turned off.

(By default, all lights are turned on and so there is no need to call this function.)

### Example

```
#include <NINJA.H>

.....
NJS_LIGHT light[5];
njInitSystem();
njCreateLight(&light[0], NJD_DIR_LIGHT);
njCreateLight(&light[1], NJD_POINT_LIGHT);
njCreateLight(&light[2], NJD_SPOT_LIGHT);
njCreateLight(&light[3], NJD_SPOT_LIGHT);
njCreateLight(&light[4], NJD_SPOT_LIGHT);

.....
/* Turn off the three light sources */
njLightOff(&light[0]);
njLightOff(&light[2]);
njLightOff(&light[3]);

.....
/* Turn on all light sources */
njLightAllOn();
```

### Remark

The relationships of njCreateLight/njDeleteLight and njLightOn/njLightOff functions are related to, for example, installing light bulb/removing light bulb and turning on installed lights/turning off installed lights.

### Related Items

njCreateLight  
njLightOff

## **njLightAllOff**

This function does not reflect all the lights set at njCreateLight (i.e. it turns off the lights).

---

### **Syntax**

```
#include <Ninja.h>
void  njLightAllOff( void )
NJS_LIGHT  *ptr
```

### **Parameters**

None

### **Return Value**

None

### **Error**

None

### **Description**

This function turns of all the lights that were set.

However, after the lights are turned off, they can be turned on again by calling njLightOn again.

### Example

```
#include <NINJA.H>

.....
NJS_LIGHT light[5];
njInitSystem();
njCreateLight(&light[0], NJD_DIR_LIGHT);
njCreateLight(&light[1], NJD_POINT_LIGHT);
njCreateLight(&light[2], NJD_SPOT_LIGHT);
njCreateLight(&light[3], NJD_SPOT_LIGHT);
njCreateLight(&light[4], NJD_SPOT_LIGHT);

.....
/* Turn off all light sources */
njLightAllOff();
```

### Remark

The relationships of njCreateLight/njDeleteLight and njLightOn/njLightOff functions are related to, for example, installing light bulb/removing light bulb and turning on installed lights/turning off installed lights.

### Related Items

njCreateLight  
njLightOn

## **njLightOff**

Deactivates a light created by njCreateLight (turns the light off).

---

### **FORMAT**

```
#include <NINJA.H>
void njLightOff( *ptr )
NJS_LIGHT *ptr
```

### **PARAMETERS**

**\*ptr**

light pointer

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Extinguish a set light.
- Afterwards, the light can be turned back on by making another call to njLightOn.

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_DIR_LIGHT);
.....
/* Extinguish parallel light source */
njLightOff(&light);
.....
```

### **NOTES**

Metaphorically, njCreateLight and njDeleteLight have the same function as installing or removing a light bulb, while njLightOn and njLightOff have the function of turning on or off the light switch.

### **RELATED TOPICS**

njCreateLight()

njLightOn()

## **njLightOn**                      Activates a light created by njCreateLight (turns the light on).

---

### **FORMAT**

```
#include <NINJA.H>
void njLightOn( *ptr )
NJS_LIGHT *ptr
```

### **PARAMETERS**

**\*ptr**

light pointer

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Turns on an extinguished light. (Lights are on by default, so this function does not need to be called unless a light is extinguished.)

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_DIR_LIGHT);
.....
/* Extinguish parallel light source */
njLightOff(&light);
.....
/* Turn on parallel light source */
njLightOn(&light);
```



## NOTES

- Metaphorically, `njCreateLight` and `njDeleteLight` have the same function as installing or removing a light bulb, while `njLightOn` and `njLightOff` have the function of turning on or off the light switch.

## RELATED TOPICS

`njCreateLight`

`njLightOff`

## **njMultiLightMatrix**

Multiplies a matrix with a light matrix.

---

### **FORMAT**

```
#include <NINJA.H>
void njMultiLightMatrix( *ptr, *m )
NJS_LIGHT *ptr
NJS_MATRIX *m
```

### **PARAMETERS**

**\*ptr**

light pointer

**\*m**

matrix pointer

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Multiplies a light matrix defined with njCreateLight by another matrix specified by the user. This results in calculation of the light source position and direction.

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;
NJS_MATRIX m;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();
```

```
/* Multiply by matrix m */  
njMultiLightMatrix(&light, &m);
```

### NOTES

- Before calling this function, the light to be modified must be specified by `njCreateLight` and `njClearMatrix` must be called.
- Also, do not apply any scaling factor to the matrix.
- The order of multiplication is as follows.

```
njMultiMatrix( light matrix , user-specified matrix );
```

### RELATED TOPICS

`njMultiMatrix()`

`njCreateLight()`

## **njRotateLightX**

Rotates a light matrix around the X axis.

---

### **FORMAT**

```
#include <NINJA.H>
void njRotateLightX( *ptr, angx )
NJS_LIGHT *ptr
Angle angx
```

### **PARAMETERS**

**\*ptr**

light pointer

**angx**

angle of rotation around X axis

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Rotates a light matrix defined with njCreateLight around the X axis.
- This results in calculation of the light source position and direction.

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Rotate 90 degrees around X axis */
```

```
njRotateLightX(&light, NJM_RAD_ANG(NJD_PI/2.f));
```

### NOTES

- Before calling this function, the light to be modified must be specified by `njCreateLight` and `njClearMatrix` must be called.

### RELATED TOPICS

`njCreateLight`

## **njRotateLightXYZ**

Rotates a light matrix around the X, Y, and Z axes.

---

### **FORMAT**

```
#include <NINJA.H>
void njRotateLightXYZ( *ptr, angx, angy, angz )
NJS_LIGHT *ptr
Angle angx
Angle angy
Angle angz
```

### **PARAMETERS**

**\*ptr**

light pointer

**angx**

angle of rotation around X axis

**angy**

angle of rotation around Y axis

**angz**

angle of rotation around Z axis

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Rotates a light matrix defined with njCreateLight around the X, Y, and Z axes, in that sequence.
- This results in calculation of the light source position and direction.

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Rotate 90 degrees around X axis,
        60 degrees around Y axis,
        and 30 degrees around Z axis */
njRotateLightXYZ(&light,
                                     NJM_RAD_ANG(NJD_PI/2.f),
                                     NJM_RAD_ANG(NJD_PI/3.f),
                                     NJM_RAD_ANG(NJD_PI/6.f) );
```

## NOTES

- Before calling this function, the light to be modified must be specified by njCreateLight and njClearMatrix must be called.

## RELATED TOPICS

[njCreateLight](#)

## **njRotateLightY**

Rotates a light matrix around the Y axis.

---

### **FORMAT**

```
#include <NINJA.H>
void njRotateLightY( *ptr, angy )
NJS_LIGHT *ptr
Angle angy
```

### **PARAMETERS**

**\*ptr**

light pointer

**angy**

angle of rotation around Y axis

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Rotates a light matrix defined with njCreateLight around the Y axis.
- This results in calculation of the light source position and direction.

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Rotate 60 degrees around Yaxis */
```



```
njRotateLightY(&light, NJM_RAD_ANG(NJD_PI/3.f));
```

### NOTES

Before calling this function, the light to be modified must be specified by `njCreateLight` and `njClearMatrix` must be called.

### RELATED TOPICS

`njCreateLight`

## **njRotateLightZ**

Rotates a light matrix around the Z axis.

---

### **FORMAT**

```
#include <NINJA.H>
void njRotateLightZ( *ptr, angz )
NJS_LIGHT *ptr
Angle angz
```

### **PARAMETERS**

**\*ptr**

light pointer

**angz**

angle of rotation around z axis

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Rotates a light matrix defined with njCreateLight around the Z axis.
- This results in calculation of the light source position and direction.

### **EXAMPLE**

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Rotate 30 degrees around Z axis */
njRotateLightZ(&light, NJM_RAD_ANG(NJD_PI/6.f));
```

## NOTES

- Before calling this function, the light to be modified must be specified by `njCreateLight` and `njClearMatrix` must be called.

## RELATED TOPICS

`njCreateLight()`

## **njSetLightAngle**

Sets the limit angle of a light created with njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetLightAngle( *ptr, iang, oang )
NJS_LIGHT *ptr
NJS_Angle iang
NJS_Angle oang
```

### **PARAMETERS**

**\*ptr**

light pointer

**iang**

limiting inside cone angle (circular angle)

**oang**

limiting outside cone angle (circular angle)

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets the limit angle of spotlight and specular light sources defined with njCreateLight.
- An example would be a spotlight used to illuminate a stage.
- The light displayed on the stage is circular, but the light is attenuated gradually outside the light cone, rather than becoming dark abruptly.
- The iang parameter of this function sets the angle of the edge of the cone, or the angle at which attenuation of light begins.
- The angle through which attenuation takes place can never exceed 90 degrees, but calculation of attenuation close to 90 degrees becomes unmeaningful as the beam of the spotlight is tightened.
- The angle at which calculation of attenuation is stopped (cut off)--i.e., the angle at which light disappears – is set by the oang parameter.

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_SPEC_POINT);
.....
/*
 * Set inner limit angle of specular point light source to 30 degrees
 * and output limit angle to 60 degrees
 */
njSetLightAngle(&light, NJM_RAD_ANG(NJD_PI/6.f), NJM_RAD_ANG(NJD_PI/3.f));
```

## NOTES

Before using this function, the light source must first be defined using `njCreateLight`.

## RELATED TOPICS

`njCreateLight()`

`njSetLightRange()`

## **njSetLightColor**      Sets the color of light defined by njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetLightColor( *ptr red, green, blue )
NJS_LIGHT *ptr
Float red
Float green
Float blue
```

### **PARAMETERS**

**\*ptr**

light pointer

**red**

red light source

**green**

green light source

**blue**

blue light source

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets the 3-color components of lighting.
- For each component, 100% is denoted by 1.f.
- This is the default for all components.

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_DIR_LIGHT);
.....
/* Set the color of a parallel light source to 255,0,0 (red) */
njSetLightColor(&light, 1.f, 0.f, 0.f);
```

## NOTES

- Before using this function, the light source must first be defined using `njCreateLight`.

## RELATED TOPICS

`njCreateLight()`

## **njSetLightDirection**

Sets the directrion of light defined by njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetLightDirection( *ptr, dx, dy, dz )
NJS_LIGHT *ptr
Float dx
Float dy
Float dz
```

### **PARAMETERS**

**\*ptr**

light pointer

**dx**

x coordinate of light source

**dy**

y coordinate of light source

**dz**

z coordinate of light source

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets the direction of the light beam as vector components. (The default direction is that of the z axis.)



## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_DIR_LIGHT);
.....
/* Set direction vector of parallel light source beam to (1,0,0) */
njSetLightDirection(&light, 1.f, 0.f, 0.f);
```

## NOTES

- Before using this function, the light source must first be defined using `njCreateLight`.
- The vector set is normalized.

## RELATED TOPICS

`njCreateLight()`

## **njSetLightIntensity**

Sets the intensity of light defined using njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetLightIntensity( *ptr, spc, dif, amb )
NJS_LIGHT *ptr
Float spc
Float dif
Float amb
```

### **PARAMETERS**

**\*ptr**

light pointer

**spc**

specular light intensity

**dif**

diffuse light intensity

**amb**

ambient light intensity

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- In the Ninja light model, light consists of three elements: specular light (highlighting), diffuse light (normal light), and ambient light (background light).
- This function sets the intensity (strength) of each of these elements.
- Ordinarily, settings are made in the range from 0.f to 1.f; however, values in excess of 1.f can be used according to conditions.

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_SPEC_DIR);
.....
/* Set intensity of specular diffuse parallel light source to 0.5f */
njSetLightIntensity(&light, 0.5f, 0.5f, 0.f);
```

## NOTES

Before using this function, the light source must first be defined using `njCreateLight`. It is not mandatory that intensity of each element of the specified light be set, in which case the intensity calculation is ignored. For instance, in the example above, setting 0.5f instead of 0.f as the 4th parameter of `njSetLightIntensity` will have no effect on this light source.

## RELATED TOPICS

`njCreateLight()`

## **njSetLightLocation**

Sets the location of light defined by njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetLightLocation( *ptr, px, py, pz )
NJS_LIGHT *ptr
Float px
Float py
Float pz
```

### **PARAMETERS**

**\*ptr**

light pointer

**px**

x coordinate of light source location

**py**

y coordinate of light source location

**pz**

z coordinate of light source location

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

Sets the location of the light source. (The default location is the origin.)

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);
.....

/* Set point light source location to (10,0,-10) */
njSetLightLocation(&light, 10.f, 0.f, -10.f);
```

## NOTES

- Before using this function, the light source must first be defined using `njCreateLight`.

## RELATED TOPICS

`njCreateLight()`

## **njSetLightRange**      Sets the limit distance of a light created with njCreateLight.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetLightRange( *ptr, nrang, frang )
NJS_LIGHT *ptr
Float nrang
Float frang
```

### **PARAMETERS**

**\*ptr**

light pointer

**nrang**

forward limit distance

**frang**

backward limit distance

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets the limiting distance of spotlighting or other lighting created by njCreateLight that requires distance calculation. Brightness increases as the light source approaches a point, but there is a limit to how far this can be expressed on the monitor screen. Accordingly, the Ninja light model continually calculates the maximum intensity of a light source when the distance to that source is reduced below a certain amount. This limit is set as the forward limit distance.
- Further, light is attenuated as distance from the light source increases, but it is not essential that calculations be performed for all distances. The point at which calculation of attenuation starts being abandoned (cut off; i.e, the distance at which light disappears) is set as the backward limit distance.



**NOTE:** Although nrang is set to 1.f by default, this setting should take into account the size of the model itself, as well as the distance between models.

---

## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_SPEC_POINT);
.....
/*
 * Set the forward limit distance of a specular point light source to 20.f
 * and the backward limit distance to 80.f.
 */
njSetLightRange(&light, 20.f, 80.f);
```

## NOTES

- Before using this function, the light source must first be defined using `njCreateLight`.

## RELATED TOPICS

`njCreateLight()`

`njSetLightAngle()`

## **njSetUserLight**                      Assigns a user-defined light function to a light.

---

### **FORMAT**

```
#include <NINJA.H>
void njSetUserLight( *ptr, func )
NJS_LIGHT *ptr
NJF_LIGHT_FUNC func
```

### **PARAMETERS**

**\*ptr**

light pointer

**func**

name of callback function (pointer)

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

Sets a user-defined light function by assigning a light source created with njCreateLight to NJD\_USER\_LIGHT.



## EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

void
func(NJS_ARGB* argb, NJS_POINT3* pnt, NJS_VECTOR* nml, NJS_LIGHT* lt)
{
    .....
                                /*
                                *   Example of inner sum of polygon normal line
                                *   and light beam direction
                                */

                                deg = - nml->x * NJM_LIGHT_VECTOR(lt).x
                                - nml->y * NJM_LIGHT_VECTOR(lt).y
                                - nml->z * NJM_LIGHT_VECTOR(lt).z;

    .....
    /* argb */

                                argb->a = deg * NJM_LIGHT_DIF(lt).a;
                                argb->r = deg * NJM_LIGHT_DIF(lt).r;
                                argb->g = deg * NJM_LIGHT_DIF(lt).g;
                                argb->b = deg * NJM_LIGHT_DIF(lt).b;

}

/* In main routine */
.....
njInitSystem();
njCreateLight(&light,  NJD_USER_LIGHT);

/* Set user function func to light lt */
njSetUserLight (&light, func);
.....
```

## NOTES

The NJD\_USER\_LIGHT light must be selected with njCreateLight.

Light functions definable by the user and macros that are used thereby are as follows.

## 1.Callback functions

<FORMAT>

```
func(rgba, pnt, nml, ptr)
                                NJS_RGBA    *rgba
                                NJS_POINT3    *pnt
                                NJS_VECTOR    *nml
                                NJS_LIGHT     *ptr
```

<Parameters>

**\*rgba**

Pointer after calculation

**\*pnt**

Polygon position vector pointer

**\*nml**

Polygon normal line vector pointer

**\*ptr**

Light pointer set by njCreateLight

## 2.Macros (with NJS\_LIGHT \*ptr)

**NJM\_LIGHT\_VECTOR(ptr)**

Current light direction: NJS\_VECTOR. This is referenced as

NJM\_LIGHT\_VECTOR(ptr).x.

**NJM\_LIGHT\_POINT(ptr)**

Current light position: NJS\_POINT3. This is referenced as

NJM\_LIGHT\_POINT(ptr).x.

**NJM\_LIGHT\_AMB(ptr)**

Ambient light intensity: NJS\_RGBA. This normally has a value in the range 0 to 1.f; however, values greater than 1.f are possible. This is referenced as

NJM\_LIGHT\_AMB(ptr).r.

**NJM\_LIGHT\_DIF(ptr)**

Diffuse light intensity: NJS\_ARGB. This normally has a value in the range 0 to 1.f; however, values greater than 1.f are possible. This is referenced as

NJM\_LIGHT\_DIF(ptr).r.

**NJM\_LIGHT\_SPC(ptr)**

Specular light intensity: NJS\_ARGB. This normally has a value in the range 0 to 1.f; however, values greater than 1.f are possible. This is referenced as

NJM\_LIGHT\_SPC(ptr).r.

**NJM\_LIGHT\_EXP(ptr)**

Index: Part of the information contained in the object data; an integer. This provides the specular diffusion. The relationship to SOFTIMAGE is listed below.

<b>NINJA</b>	:	SOFTIMAGE
<b>0</b>	:	0
<b>1</b>	:	1
<b>2</b>	:	2
<b>3</b>	:	3
<b>4</b>	:	4
<b>5</b>	:	6
<b>6</b>	:	8
<b>7</b>	:	12
<b>8</b>	:	16
<b>9</b>	:	24
<b>10</b>	:	32
<b>11</b>	:	48
<b>12</b>	:	64
<b>13</b>	:	96
<b>14</b>	:	128
<b>15</b>	:	192
<b>16</b>	:	256

### **NJM\_LIGHT\_COLOR(ptr)**

Light color: NJS\_ARGB. This provides the light color.

### **NJM\_LIGHT\_INIT\_VECTOR(ptr)**

Light direction before matrix calculation: NJS\_VECTOR. This is the initial direction before matrix calculation.

### **NJM\_LIGHT\_INIT\_POINT(ptr)**

Light position before matrix calculation: NJS\_POINT3. This is the initial position before matrix calculation.

### **NJM\_LIGHT\_MATRIX(ptr)**

Light position before matrix calculation. This is the light matrix.

## RELATED TOPICS

[njCreateLight\(\)](#)

## njTranslateLight

Applies a matrix that gives parallel translation along each axis.

---

### FORMAT

```
#include <NINJA.H>
void njTranslateLight( *ptr, tx, ty, tz )
NJS_LIGHT *ptr
Float tx
Float ty
Float tz
```

### PARAMETERS

**\*ptr**

Light pointer

**tx**

Move in X-axis direction

**ty**

Move in Y-axis direction

**tz**

Move in Z-axis direction

### RETURN VALUE

None

### ERROR VALUE

None

### FUNCTION

- Translates light matrix registered by njCreateLight with elements x, y, and z.
- Location and direction of light source will be calculated.

### EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Translate with elements of (10,0,-10) */
njTranslateLight(&light, 10.f, 0.f, -10.f);
```

### NOTES

Light must be specified by `njCreateLight`, and `njClearMatrix` must be called beforehand.

### RELATED TOPICS

`njCreateLight()`

## **njTranslateLightV**

Moves a light matrix laterally according to a directional vector.

---

### **FORMAT**

```
#include <NINJA.H>
void njTranslateLightV( *ptr, *vctr )
NJS_LIGHT *ptr
NJS_VECTOR *vctr
```

### **PARAMETERS**

**\*ptr**

light pointer

**\*vctr**

pointer to directional vector

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Uses a directional vector to move a light matrix created with njCreateLight. This results in calculation of the position and direction of the light source.

### EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;
NJS_VECTOR vector;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Move by vector */
njTranslateLightV(&light, &vector);
```

### NOTES

- Before calling this function, the light source must be specified by `njCreateLight` and `njClearMatrix` must be called.

### RELATED TOPICS

`njCreateLight()`



## **njUnitLightMatrix**

Unitizes a light matrix.

---

### **FORMAT**

```
#include <NINJA.H>
void njUnitLightMatrix( *ptr )
NJS_LIGHT *ptr
```

### **PARAMETERS**

**\*ptr**

light pointer

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Unitizes a light matrix created with njCreateLight.
- This results in calculation of the position and direction of the light source.

### EXAMPLE

```
#include <NINJA.H>
.....

NJS_LIGHT light;

njInitSystem();
njCreateLight(&light, NJD_POINT_LIGHT);

.....
njClearMatrix();

/* Unitizes the light matrix */
njUnitLightMatrix(&light);
```

### NOTES

- Before using this function, the light must be specified with `njCreateLight` and `njClearMatrix` must be called.

### RELATED TOPICS

`njCreateLight()`



## *8. Scroll Functions*

---

### **Contents**

njDrawScroll ..... Draws a 2D scroll surface. .... NLS-240

# **njDrawScroll**

Draws a 2D scroll surface.

## **FORMAT**

```
#include <Ninja.h>
void njDrawScroll( *scl )
NJS_SCROLL *scl
```

## **PARAMETERS**

**\*scl**

pointer to scroll structure

## **RETURN VALUE**

None

## **FUNCTION**

- Draws a 2D scroll structure in the clip screen.

**EXAMPLE**

```
Uint32 map[4][5];
NJS_TEXLIST texlist;

NJS_TEXMEMLIST *texmemlist;

/* Cell definition */
/* Prepare to load 5 textures */
texmemlist = malloc(sizeof(NJS_TEXMEMLIST)*5);
njInitTexture(texmemlist,5);
njLoadTexture(&texlist);

/* The scroll function does not require calling njSetTexture or njSetTextureNum */

/* Set the scroll structure */
scl.mapw = 5; /*
Number of map cells horizontally */
scl.maph = 4; /*
Number of map cells vertically */
scl.sx = 1.0f; /*
Scale horizontally */
scl.sy = 1.0f; /*
Scale vertically */
scl.spx = 320.f; /*
Scale center x coordinate */
scl.spy = 240.f; /*
Scale center y coordinate */
scl.list = &texlist; /*
Texture list */
scl.map = (Uint32*)map; /* Map
array */
scl.mflag = ON; /*
Matrix flag */
scl.sflag = ON; /*
Scale flag */
scl.pr = -100.f; /*
Priority -1.f (front) to -65535.f (rear) */
scl.px = 0.f; /*
Movement x coordinate */
scl.py = 0.f; /*
Movement y coordinate */
scl.bx = 0.f; /*
Start of map write x coordinate */
scl.by = 0.f; /*
Start of map write y coordinate */
scl.cx = 320.f; /*
Rotation center x coordinate */
scl.cy = 240.f; /*
Rotation center y coordinate */
scl.clip[0].x = 0.f; /*
Upper left clip x coordinate */
scl.clip[0].y = 0.f; /*
Upper left clip y coordinate */
```

```
scl.clip[1].x = 640.f; /*
Lower right clip x coordinate */
scl.clip[1].y = 480.f; /*
Lower right clip y coordinate */
scl.sclc.argb.a = 255; /*
Globally applied A*/
scl.sclc.argb.r = 255; /*
Globally applied R*/
scl.sclc.argb.g = 255; /*
Globally applied g*/
scl.sclc.argb.b = 255; /*
Globally applied B */
scl.colmode = NJD_COLOR_MODE_TEXTURE; /* Set
color mode */
```

## NOTES

Note that color mode has been changed.

Clipping function has been deleted.

For more detailed explanation on making map or texture list, see Scroll Guide.

With Scroll sample

## RELATED TOPICS

`njInitTexture()`

`njLoadTexture()`



## 9. Modeling Functions

---

### Contents

njSimpleDrawObject .....	Draws objects. ....	NLS-243
njControl3D.....	Controls the drawing surface for a 3D object. ....	NLS-244
njDrawModel.....	Draws a model. ....	NLS-248
njDrawObject .....	Draws an object. ....	NLS-250
njFastDrawModel .....	Draws Models. ....	NLS-252
njFastDrawObject .....	Draws objects. ....	NLS-254
njInit3D.....	Initializing 3D system. ....	NLS-256
njSetConstantAttr .....	Sets model attributes. ....	NLS-257
njSetConstantMaterial .....	Sets model material data. ....	NLS-259
njSetDepthQueue .....	Sets depth queue. ....	NLS-261
njSimpleDrawModel.....	Draws models. ....	NLS-263
njSimpleDrawObject .....	Draws objects. ....	NLS-264

## **njControl3D**

Controls the drawing surface for a 3D object.

---

### **FORMAT**

```
#include <Ninja.h>
void    njControl3D( flag )
Uint32  flag
```

### **PARAMETERS**

**flag**

control parameters

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Controls the drawing surface for models and objects.
- Parameters that can be set are as follows.

#### **NJD\_CONTROL\_3D\_DISP\_AXIS**

Draws axes (1 long) at the origin for each model.

X axis:blue, Y axis:green, Z axis:red

#### **NJD\_CONTROL\_3D\_NO\_CLIP\_CHECK**

Specifies not to perform clipping check when drawing models.

With this parameter set, drawing efficiency will be up if model drawing is within a screen completely.

#### **NJD\_CONTROL\_3D\_CONSTANT\_ATTR**

Allows to change model attribute. Changing parameters should be specified with

`njSetConstantAttr()`.

#### **NJD\_CONTROL\_3D\_CONSTANT\_MATERIAL**

Changes material data of models to fixed value. Changing parameters should be specified with `njSetConstantMaterial()`.



**NJD\_CONTROL\_3D\_OFFSET\_MATERIAL**

Supplements material data of models. Supplementing parameters should be specified with `njSetConstantMaterial()`.

**NJD\_CONTROL\_3D\_DEPTH\_QUEUE**

Validates depth queue. Depth queue here means to change degree of transparency accommodating with distance.

Depth queue parameters should be specified with `njSetDepthQueue()`.

**NJD\_CONTROL\_3D\_VERTEX\_OFFSET**

Moves vertex data in normal line direction.

The offset to move is specified by `_nj_vertex_offset_ variable`.

**NJD\_CONTROL\_3D\_MODEL\_CLIP**

Validates clipping by model.

**NJD\_CONTROL\_3D\_CONSTANT\_TEXTURE\_MATERIAL**

Makes texture material color white.

In default, it uses material color.

**EXAMPLE**

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitMatrix( matrix, 128 );
njInit3D( vbuf, 4096 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

njControl3D( NJD_CONTROL_3D_NO_CLIP_CHECK | NJD_CONTROL_3D_DISP_AXIS );

while(1) {
    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();
}
```

## NOTES

## RELATED TOPICS

`njInit3D()`

`njDrawModel()`

`njDrawObject()`

`njSetConstantAttr()`

`njSetConstantMaterial()`

`njSetTextureMode()`

`njDrawAlphaBuf()`

`njSetDepthQueue()`

## **njDrawModel**

Draws a model.

---

### **FORMAT**

```
#include <Ninja.h>
void      njDrawModel( model )
NJS_MODEL *model
```

### **PARAMETERS**

**model**

pointer to model structure

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Draws the model data after converting coordinates for the current matrix.
- The drawing method can be controlled with njControl3D().

### **EXAMPLE**

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitMatrix( matrix, 128 );
njInit3D( vbuf, 4096 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

while(1) {
    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njDrawModel( model );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();
}
```

## NOTES

## RELATED TOPICS

`njInit3D()`

`njDrawObject()`

`njControl3D()`

`njSetConstantAttr()`

`njSetConstantMaterial()`

`njSetTextureMode()`

`njDrawAlphaBuf()`

`njSetDepthQueue()`

## **njDrawObject**

Draws an object.

---

### **FORMAT**

```
#include <Ninja.h>
void      njDrawObject( *object )
NJS_OBJECT *object
```

### **PARAMETERS**

**\*object**

pointer to object structure

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

Draws the object (model data with parent-child hierarchy) after converting coordinates for the current matrix.

The drawing method can be controlled with njControl3D().

### EXAMPLE

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitMatrix( matrix, 128 );
njInit3D( vbuf, 4096 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

while(1) {

    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();

}
```

### NOTES

### RELATED TOPICS

- [njInit3D\(\)](#)
- [njDrawModel\(\)](#)
- [njControl3D\(\)](#)
- [njSetConstantAttr\(\)](#)
- [njSetConstantMaterial\(\)](#)
- [njSetTextureMode\(\)](#)
- [njDrawAlphaBuf\(\)](#)
- [njSetDepthQueue\(\)](#)

# **njFastDrawModel**

Draws Models.

---

## **FORMAT**

```
#include <Ninja.h>
void      njFastDrawModel( model )
NJS_MODEL *model
```

## **PARAMETERS**

**model**

model structure pointer

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Performs coordinate conversion for model data at the current matrix, and draws it.
- Drawing method can be controlled by njControl3D().
- Drawing ability is higher than njDrawModel().
- However, only white is allowed as a light color and depth queue cannot function.

### EXAMPLE

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitVertexBuffer( 500000, 0, 10000, 0 );
njInitMatrix( matrix, 128, 0 );
njInit3D( vbuf, 4096 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

while(1) {

    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njFastDrawModel( model );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();

}
```

### NOTES

### RELATED TOPICS

- [njInit3D\(\)](#)
- [njDrawObject\(\)](#)
- [njControl3D\(\)](#)
- [njSetConstantAttr\(\)](#)
- [njSetConstantMaterial\(\)](#)
- [njSetTextureMode\(\)](#)
- [njDrawAlphaBuf\(\)](#)
- [njSetDepthQueue\(\)](#)



## **njFastDrawObject**

Draws objects.

---

### **FORMAT**

```
#include <Ninja.h>
void  njFastDrawObject( object1 )
NJS_OBJECT  *object
```

### **PARAMETERS**

**object**

pointer for the object structure

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Performs coordinate conversion for object (model data with parent/child hierarchy) at the current matrix, and draws it.
- Drawing method can be controlled by njControl3D().
- Drawing ability is higher than njDrawObject().
- However, only white is allowed as a light color and depth queue cannot function.

### EXAMPLE

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitVertexBuffer( 500000, 0, 10000, 0 );
njInitMatrix( matrix, 128, 0 );
njInit3D( vbuf, 4096 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

while(1) {

    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njFastDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();

}
```

### NOTES

### RELATED TOPICS

- [njInit3D\(\)](#)
- [njDrawModel\(\)](#)
- [njControl3D\(\)](#)
- [njSetConstantAttr\(\)](#)
- [njSetConstantMaterial\(\)](#)

## njInit3D

Initializing 3D system.

---

### FORMAT

<code>#include</code>	<code>&lt;Ninja.h&gt;</code>
<code>void</code>	<code>njInit3D( vbuf, vn )</code>
<code>NJS_VERTEX_BUF</code>	<code>*vbuf</code>
<code>Int</code>	<code>vn</code>

### PARAMETERS

**vbuf**

pointer to vertex calculation working buffer

**vn**

size of vertex calculation working buffer

### RETURN VALUE

None

### ERROR VALUE

None

### FUNCTION

- Initializes the work buffer used for model drawing.
- The vertex computation work buffer is used for temporary storage of the model data vertex computation results.  
If multiple models are drawn, the buffer size used can only be as large as the largest of the models.

### EXAMPLE

```
#define VERTEX_MAX 1024
NJS_VERTEX_BUF vbuf[VERTEX_MAX];
njInit3D( vbuf, VERTEX_MAX );
```

### NOTES

### RELATED TOPICS

`njDrawModel()`

`njDrawObject()`

`njControl3D()`

`njDrawAlphaBuf()`

## **njSetConstantAttr**

Sets model attributes.

### **FORMAT**

```
#include <Ninja.h>
void      njSetConstantAttr( and_attr, or_attr )
Uint32    and_attr
Uint32    or_attr
```

### **PARAMETERS**

**and\_attr**

AND attribute pattern

**or\_attr**

OR attribute pattern

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Specifies the model attributes.
- These are specified as an AND pattern or OR pattern which is applied to the original data. The attributes are enabled when NJD\_CONTROL\_3D\_CONSTANT\_ATTR is specified with njControl3D.

## EXAMPLE

This changes a polygon with texture to a monocolored polygon and displays it flat.

```
njInitSystem( NJD_RESOLUTION_640x480, 1 );
njInitMatrix( matrix, 128 );
njInit3D( vbuf, 4096, abuf, zbuf, 32 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

njControl3D( NJD_CONTROL_3D_CONSTANT_ATTR );
njSetConstantAttr( NJD_FLAG_USE_TEXTURE, NJD_FLAG_USE_FLAT );

while(1) {
    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();
}
```

## NOTES

## RELATED TOPICS

[njDrawModel\(\)](#)

[njDrawObject\(\)](#)

[njControl3D\(\)](#)

## **njSetConstantMaterial**

Sets model material data.

### **FORMAT**

```
#include <Ninja.h>
void      njSetConstantMaterial( argb )
NJS_ARGB  *argb
```

### **PARAMETERS**

**argb**

material data

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Specifies a model's material data.
- The material data becomes valid when NJD\_CONTROL\_3D\_CONSTANT\_MATERIAL or NJD\_CONTROL\_3D\_OFFSET\_MATERIAL is specified by njControl3D.
- When NJD\_CONTROL\_3D\_CONSTANT\_MATERIAL is specified, the model is drawn with the specified material data. When NJD\_CONTROL\_3D\_OFFSET\_MATERIAL is specified, the model is drawn after adding the specified material data to the material of the model data.
- This data is also used for color calculation when drawing sprites.

## EXAMPLE

The model is drawn wholly transparent.

```
njInitSystem( NJD_RESOLUTION_640x480, 1 );
njInitMatrix( matrix, 128 );
njInit3D( vbuf, 4096, abuf, zbuf, 32 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

njControl3D( NJD_CONTROL_3D_OFFSET_MATERIAL );
argb.a = -128.0f;
argb.r = 0.f;
argb.g = 0.f;
argb.b = 0.f;
njSetConstantMaterial( &argb );

while(1) {
    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();
}
```

## NOTES

## RELATED TOPICS

- `njDrawModel()`
- `njDrawObject()`
- `njControl3D()`
- `njDrawSprite2D()`
- `njDrawSprite3D()`

## **njSetDepthQueue**

Sets depth queue.

---

### **FORMAT**

```
#include <Ninja.h>
void  njSetDepthQueue( near, far )
Float  near
Float  far
```

### **PARAMETERS**

**near**

Sets depth queue.

**far**

back Z value

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets depth queue of model drawing.
- Enables when specifies NJD\_CONTROL\_3D\_DEPTH\_QUEUE for njControl3D.
- Depth queue changes the degree of transparency from the front towards the back.
- The part which is front or “near” does not become transparent. The part which is back or “far” becomes transparent..



## EXAMPLE

```
njInitSystem( NJD_RESOLUTION_640x480, 1 );
njInitMatrix( matrix, 128 );
njInit3D( vbuf, 4096, abuf, zbuf, 32 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT );
njSetView( &view );

njControl3D( NJD_CONTROL_3D_DEPTH_QUEUE );
njSetDepthQueue( -9000.f, -10000.f );
njClipZ( -10.f, -10000.f );

while(1) {
    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();
}
```

The degree of transparency is changed when Z value is -9000 thru -10000.

If the far values for both Z clip and depth queue are set the same, clipping will be done when it becomes completely transparent.

## NOTES

## RELATED TOPICS

[njDrawModel\(\)](#)

[njDrawObject\(\)](#)

[njControl3D\(\)](#)

# njSimpleDrawModel

Draws models.

## FORMAT

```
#include <Ninja.h>
void njSimpleDrawModel( model )
NJS_MODEL *model
```

## PARAMETERS

**model**

Pointer to model structure

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- This function converts the coordinates for model data according to the current matrix and then draws the model.
- The drawing method can be controlled via njControl3D().
- The drawing performance of this routine is better than that of njFastDrawModel().
- However, the light that is registered first is regarded as a parallel light source, so that only the direction of the light source has any effect; the color and intensity of the light source are not reflected in the drawing.
- Furthermore, this function does not support depth cues.

## EXAMPLE

## NOTES

## RELATED TOPICS

## **njSimpleDrawObject**

Draws objects.

---

### **FORMAT**

```
#include <Ninja.h>

void njSimpleDrawObject( objectI )

NJS_OBJECT *object
```

### **PARAMETERS**

objectPointer to object structure

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- This function converts the coordinates for an object (model data with a parent-child hierarchy) according to the current matrix and then draws the object.
- The drawing method can be controlled via njControl3D().
- The drawing performance of this routine is better than that of njFastDrawObject().
- However, the light that is registered first is regarded as a parallel light source, so that only the direction of the light source has any effect; the color and intensity of the light source are not reflected in the drawing.
- Furthermore, this function does not support depth cues.

### EXAMPLE

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );
njInitVertexBuffer( 500000, 0, 10000, 0 );
njInitMatrix( matrix, 128, 0 );
njInit3D( vbuf, 4096 );
njInitView( &view );
njCreateLight( &light, NJD_DIR_LIGHT ); njSetView( &view );

while(1) {
    njClearMatrix();
    njTranslate( NULL, 0.f, 0.f, -10.f );
    njRotateXYZ( NULL, xx,yy,zz );
    njSimpleDrawObject( object );
    xx += 257;
    yy += 179;
    zz += 193;
    njWaitVSync();
}
```

### NOTES

### RELATED TOPICS

- [njInit3D\(\)](#)
- [njDrawModel\(\)](#)
- [njControl3D\(\)](#)
- [njSetConstantAttr\(\)](#)
- [njSetConstantMaterial\(\)](#)



## 10. View Functions

---

### Contents

njCalcScreen.....	Projects points in 3D space onto the screen, then finds the screen coordinates to which the points are projected. ....	NLS-269
njClip2D.....	Specifies the drawing area on the screen. ....	NLS-271
njClipZ .....	Specifies the limit values of near clipping and far clipping. ....	NLS-272
njForwardViewAbsolute .....	Moves the view location in the direction of the view. (Absolute Move). ....	NLS-273
njForwardViewRelative.....	Moves the view location along the viewline. (Relative Move). ....	NLS-275
njInitView .....	Initializes the view. ....	NLS-277
njLookAtView .....	Changes the view direction towards point (x, y, z). ....	NLS-278
njLookAtViewV .....	Changes the view direction towards point (x, y, z). ....	NLS-280
njMultiViewMatrix .....	Multiplies a view by a matrix. ....	NLS-282
njReturn2BaseView .....	Returns the current view to the base view. ....	NLS-283
njRotateViewPosXAbsolute.....	Rotates the view location around the X axis. (Absolute Rotation). ....	NLS-284
njRotateViewPosXRelative .....	Rotates the view around the X axis (Relative Rotation). ....	NLS-286
njRotateViewPosYAbsolute.....	Rotates the view location around the Y axis. (Absolute Rotation). ....	NLS-288
njRotateViewPosYRelative .....	Rotates the view around the Y axis (Relative Rotation). ....	NLS-290
njRotateViewPosZAbsolute.....	Rotates the view location around the Z axis. (Absolute Rotation). ....	NLS-292
njRotateViewPosZRelative .....	Rotates the view around the Z axis (Relative Rotation). ....	NLS-294
njRotateViewX.....	Rotates the view around the X axis (Absolute Rotation). ....	NLS-296
njRotateViewXAbsolute .....	Rotates the view location around the X axis. (Absolute Rotation). ....	NLS-298
njRotateViewXRelative.....	Rotates the view around the X axis (Relative Rotation). ....	NLS-300
njRotateViewXYZ .....	Rotates the view around the X, Y, and Z axes (Absolute Rotation). ....	NLS-302
njRotateViewXYZAbsolute.....	Rotates the line of view around the X, Y, and Z axes. (Absolute Rotation). ....	NLS-304

<b>njRotateViewXYZRelative</b> .....	Rotates the view around the X, Y, and Z axes (Relative Rotation). .....	NLS-306
<b>njRotateViewY</b> .....	Rotates the view around the Y axis (Absolute Rotation). .....	NLS-308
<b>njRotateViewYAbsolute</b> .....	Rotates the view line around the Y axis. (Absolute Rotation). .....	NLS-310
<b>njRotateViewYRelative</b> .....	Rotates the view around the Y axis (Relative Rotation). .....	NLS-312
<b>njRotateViewZ</b> .....	Rotates the view around the Z axis (Absolute Rotation). .....	NLS-314
<b>njRotateViewZAbsolute</b> .....	Rotates the view around the Z axis. (Absolute Rotation). .....	NLS-316
<b>njRotateViewZRelative</b> .....	Rotates the view around the Z axis (Relative Rotation). .....	NLS-318
<b>njSetAspect</b> .....	Sets the screen aspect ratio. ....	NLS-320
<b>njSetBaseView</b> .....	Sets the current view as the base view. ....	NLS-321
<b>njSetPerspective</b> .....	Sets the perspective in horizontal direction. ....	NLS-323
<b>njSetScreen</b> .....	Sets the screen. ....	NLS-324
<b>njSetScreenDist</b> .....	Sets the distance from the perspective to the screen. ....	NLS-325
<b>njSetView</b> .....	Specifies a user-defined view as the current view. ....	NLS-326
<b>njTranslateView</b> .....	Translates the view along the X, Y, and Z axes (Absolute Translation). ....	NLS-327
<b>njTranslateViewAbsolute</b> .....	Moves the view location along the X, Y, and Z axes. (Absolute Move). ....	NLS-329
<b>njTranslateViewRelative</b> .....	Translates the view along the X, Y, and Z axes (Relative Translation). ....	NLS-331
<b>njTranslateViewV</b> .....	Translates the view along the X, Y, and Z axes (Absolute Translation). ....	NLS-333
<b>njTranslateViewVAbsolute</b> .....	Moves the view location along the X, Y, and Z axes. (Absolute Move). ....	NLS-335
<b>njTranslateViewVRelative</b> .....	Translates the view along the X, Y, and Z axes (Relative Translation). ....	NLS-337
<b>njUnitBaseViewVector</b> .....	Converts the original view vector to a unit vector. ....	NLS-339
<b>njUnitCurrentViewVector</b> .....	Converts the view vector of the current view to a unit vector. ....	NLS-340
<b>njUnitViewMatrix</b> .....	Sets a unit matrix to the view matrix. ....	NLS-341
<b>njUnitViewVector</b> .....	Converts the view vector to a unit vector. ....	NLS-342

## njCalcScreen

Projects points in 3D space onto the screen, then finds the screen coordinates to which the points are projected.

---

### FORMAT

```
#include <Ninja.h>

int njCalcScreen(*p, *sx, *sy)
NJS_POINT3 *p
Float *sx
Float *sy
```

### PARAMETERS

**p**

point coordinates in 3D space

**sx**

x coordinate after projection

**sy**

y coordinate after projection

### RETURN VALUE

**OK**

The coordinates after projection lie within the screen's drawing area.

**NG**

The coordinates after projection fall outside of the screen's drawing area.

### FUNCTION

- Projects points in 3D space onto the screen, then finds the screen coordinates to which the points are projected.

### EXAMPLE

The following projects the point at (1000, 1000, 1000) in 3D space onto the screen, then finds the screen coordinates.

```
NJS_POINT3 p;  
Float sx, sy;  
  
p.x = 1000.f;  
p.y = 1000.f;  
p.z = 1000.f;  
  
njInitSystem(NJD_RESOLUTION_640x480, 1);  
njCalcScreen(&p, &sx, &sy);  
  
% Result %  
  
NG  
sx = 820.000000, sy = 740.000000
```

### NOTES

- The results calculated by this function reflect the screen, aspect, near clipping, far clipping, and drawing area settings.

### RELATED TOPICS

[njInitSystem\(\)](#)  
[njSetScreen\(\)](#)  
[njSetAspect\(\)](#)  
[njClip2D\(\)](#)  
[njClipZ\(\)](#)



## **njClip2D**

Specifies the drawing area on the screen.

---

### **FORMAT**

```
#include <Ninja.h>

void njClip2D(*v)
NJS_POINT2 *v
```

### **PARAMETERS**

**v**

upper right and lower left coordinates of drawing area

### **RETURN VALUE**

None

### **FUNCTION**

- Specifies the drawing area on the screen.

### **EXAMPLE**

The following specifies a drawing area with an upper right corner at (160, 240) and a lower left corner at (480, 300).

```
NJS_POINT2 p[2];

P[0].x = 160.f;
P[0].y = 240.f;
P[1].x = 480.f;
P[1].y = 300.f;

njClip2D(P);
```

### **NOTES**

Since njInitSystem() initializes the drawing area to the entire screen, this function is not needed unless the user requires a drawing area of different size.

### **RELATED TOPICS**

[njInitSystem\(\)](#)

## **njClipZ**      Specifies the limit values of near clipping and far clipping.

---

### **FORMAT**

```
#include <Ninja.h>

void njClipZ(n, f)
Float n
Float f
```

### **PARAMETERS**

**n**

near clipping limit value

**f**

far clipping limit value

### **RETURN VALUE**

None

### **FUNCTION**

- Specifies the limit values of near clipping and far clipping.
- Since the area behind the screen is represented by a negative Z axis in the Ninja coordinate system, n and f are specified as negative values.  
**(-1.0 => n > f => -65535.0)**  
Specified values of greater than -1.0 or less than -65535.0 are automatically changed to -1.0 or -65535.0, respectively.

### **EXAMPLE**

```
#define N_CLIP      -500.f
#define F_CLIP      -50000.f

njClipZ(N_CLIP, F_CLIP);
```

### **NOTES**

### **RELATED TOPICS**

## njForwardViewAbsolute

Moves the view location in the direction of the view. (Absolute Move).

---

### FORMAT

```
#include <Ninja.h>

njForwardViewAbsolute( *v, x)
NJS_VIEW *v
Float x
```

### PARAMETERS

**v**

view structure pointer

**x**

movement distance (absolute)

### RETURN VALUE

None

### FUNCTION

- Moves the view location in the direction of the view by an absolute amount.

### EXAMPLE

This example moves the view located at (0, 0, 0) and directed toward (1, 1, -1) (toward the inside of the upper right of the screen) by 10 in the direction of the view.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 0.f
#define VIEW_AVX 1.f
#define VIEW_AYV 1.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.avy = VIEW_AYV;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njUnitCurrentViewVector(&v);
njForwardViewAbsolute(&v, 10.f);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter v is NULL, this function acts on the current view.

### RELATED TOPICS

[njUnitCurrentViewVector\(\)](#)

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njForwardViewRelative**

Moves the view location along the viewline.  
(Relative Move).

---

### **FORMAT**

```
#include <Ninja.h>

njForwardViewRelative( *v,  x)
NJS_VIEW *v
Float x
```

### **PARAMETERS**

**v**

view structure pointer

**x**

movement distance (relative value)

### **RETURN VALUE**

None

### **FUNCTION**

- Moves the view location along the viewline by a relative amount.

### EXAMPLE

This example moves the view located at (0, 0, 0) and directed toward (1, 1, -1) (toward the inside of the upper right of the screen) by 10 in the direction of the view.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 0.f
#define VIEW_VX 1.f
#define VIEW_VY 1.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njUnitCurrentViewVector(&v);
njForwardViewRelative(&v, 10.f);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter `v` is `NULL`, this function acts on the current view.

### RELATED TOPICS

`njUnitCurrentViewVector()`

`njSetView()`

`njClearMatrix()`

## **njInitView**

Initializes the view.

---

### **FORMAT**

```
#include <Ninja.h>

void njInitView(*v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**  
view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Initializes the user-defined view as follows:

```
Position      (PX,PY,PZ) = (0,0, 0)                               //
origin
Orientation (VX,XY,XZ) = (0,0,-1)                                   //
toward back of screen
Tilt (roll, or tilt along the Z axis of line of sight)    //0 degrees
View matrix = Unit matrix
```

### **EXAMPLE**

```
NJS_VIEW v;
njInitView(&v);
```

### **NOTES**

- This function may be omitted when settings are made directly in the view structure.
- Structure members which must be set are:

```
px, py, pz    (view point)
vx, vy, vz    (view direction)
roll          (view tilt)
```

It is not necessary to set the view matrix.

### **RELATED TOPICS**

```
njSetView()
njClearMatrix()
```

## **njLookAtView**      Changes the view direction towards point (x, y, z).

---

### **FORMAT**

```
#include <Ninja.h>

void njLookAtView( *v, x, y, z)
NJS_VIEW *v
Float x
Float y
Float z
```

### **PARAMETERS**

**v**

view structure pointer

**x**

X coordinate of point toward which view is being directed

**y**

Y coordinate of point toward which view is being directed

**z**

Z coordinate of point toward which view is being directed

### **RETURN VALUE**

None

### **FUNCTION**

- Changes the view direction towards a point (x, y, z).



## EXAMPLE

The following changes the direction of the view located at (500, 500, 500) and directed toward (0, 0, -1) (to rear of screen) toward the origin (0, 0, 0).

```
#define VIEW_PX 500.f
#define VIEW_PY 500.f
#define VIEW_PZ 500.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njLookAtView(&v, 0.f, 0.f, 0.f);
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter `v` is `NULL`, the current view is changed.

## RELATED TOPICS

`njInitView()`

`njSetView()`

`njClearMatrix()`

## **njLookAtViewV**      Changes the view direction towards point (x, y, z).

---

### **FORMAT**

```
#include <Ninja.h>

void njLookAtViewV( *v, *p)
NJS_VIEW *v
NJS_POINT3 *p
```

### **PARAMETERS**

**v**

view structure pointer

**p**

coordinates of point to which view is to be directed

### **RETURN VALUE**

None

### **FUNCTION**

- Changes the view direction towards a point (x, y, z).

## EXAMPLE

The following changes the direction of the view located at (500, 500, 500) and directed toward (0, 0, -1) (to rear of screen) toward the origin (0, 0, 0).

```
#define VIEW_PX 500.f
#define VIEW_PY 500.f
#define VIEW_PZ 500.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;
NJS_POINT3 p;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

p.x = 0.f;
p.y = 0.f;
p.z = 0.f;

njLookAtViewRelative(&v, &p);
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter `v` is `NULL`, the current view is changed.

## RELATED TOPICS

`InitView()`

`njSetView()`

`njClearMatrix()`

# njMultiViewMatrix

Multiplies a view by a matrix.

---

## FORMAT

```
#include <Ninja.h>

void njMultiViewMatrix(*v, *m)
NJS_VIEW *v
NJS_MATRIX *m
```

## PARAMETERS

**v**

view structure pointer

**m**

matrix structure pointer

## RETURN VALUE

None

## FUNCTION

Multiplies a user-specified view by a matrix.

```
% Result %
[VM]: view matrix
[M]: matrix

[VM] = [M][VM]
(px, py, pz) = (px, py, pz)[M]
(vx, vy, vz) = (vx, vy, vz)[M]
```

## EXAMPLE

## NOTES

- When parameter v is NULL, the current matrix is used.

## RELATED TOPICS

## **njReturn2BaseView**

Returns the current view to the base view.

---

### **FORMAT**

```
#include <Ninja.h>

void njReturn2BaseView(*v);
NJS_VIEW *v
```

### **PARAMETERS**

**v**

view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Restores the location and viewline orientation of the current view to the location and orientation of the base view.

### **EXAMPLE**

This example manipulates the view in various ways, then restores the view to the base view location and viewline.

```
NJS_VIEW v;

njInitView(&v);
njTranslateViewRelative(50.f, 50.f, 50.f);
njjnjLookAtView(&v, 0.f, 0.f, 0.f);
njReturn2BaseView(&v);
njSetView(&v);
njClearMatrix();
```

### **NOTES**

- When parameter v is NULL, this function acts on the current view.

### **RELATED TOPICS**

InitView()

njSetView()

njClearMatrix()

## **njRotateViewPosXAbsolute**

Rotates the view location around the X axis. (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewPosXAbsolute(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view location around the X axis by an absolute angle.

## EXAMPLE

This example rotates the location of the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the X axis by 90 degrees, without changing the view direction.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.avy = VIEW_AY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njRotateViewPosXAbsolute(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njRotateViewPosXRelative**

Rotates the view around the X axis  
(Relative Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewPosXRelative(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (relative amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view by a relative amount around the X axis.



## EXAMPLE

Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the X axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewPosXRelative(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njRotateViewPosYAbsolute**

Rotates the view location around the Y axis. (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>
void njRotateViewPosYAbsolute(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view location around the Y axis by an absolute angle.

## EXAMPLE

This example rotates the location of the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the Y axis by 90 degrees, without changing the view direction.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AVY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.avy = VIEW_AVY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njRotateViewPosYAbsolute(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njRotateViewPosYRelative**

Rotates the view around the Y axis  
(Relative Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewPosYRelative(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (relative amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view by a relative amount around the Y axis.

## EXAMPLE

- Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the Y axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewPosYRelative(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njRotateViewPosZAbsolute**

Rotates the view location around the Z axis. (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>
void njRotateViewPosZAbsolute(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view location around the Z axis by an absolute angle.

## EXAMPLE

This example rotates the location of the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the Z axis by 90 degrees, without changing the view direction.

```
#define VIEW_APX 3000.f
#define VIEW_APY 0.f
#define VIEW_APZ 0.f
#define VIEW_AVX 0.f
#define VIEW_AY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.ay = VIEW_AY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njRotateViewPosZAbsolute(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njRotateViewPosZRelative**

Rotates the view around the Z axis  
(Relative Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewPosZRelative(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (relative amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

Rotates the view by a relative amount around the Z axis.



## EXAMPLE

Rotates the view located at (3000, 0, 0) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the Z axis.

```
#define VIEW_PX 3000.f
#define VIEW_PY 0.f
#define VIEW_PZ 0.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewPosZRelative(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njRotateViewX**

Rotates the view around the X axis (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewX(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute amount of the rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view around the X axis.

## EXAMPLE

Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the X axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njSetView(&v);
njRotateViewX(&v, NJM_DEG_ANG(90));
njClearMatrix();
```

## NOTES

- When parameter `v` is `NULL`, the current matrix is rotated.
- Views (viewpoints) can be moved and rotated by manipulating the view structure in either of two ways: by specifying transformation of view structure matrix in absolute terms, or by manipulating the position (`px`, `py`, `pz`), view direction (`vx`, `vy`, `vz`), and tilt (view roll) of each member in relative terms. This function is of the absolute type.

## RELATED TOPICS

`njSetView()`

`njClearMatrix()`

## **njRotateViewXAbsolute**

Rotates the view location around the X axis. (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>
void njRotateViewXAbsolute(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view location around the X axis by an absolute angle.

## EXAMPLE

This example rotates the location of the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the X axis by 90 degrees, without changing the view direction.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AVY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.avy = VIEW_AVY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njRotateViewXAbsolute(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

# **njRotateViewXRelative**

Rotates the view around the X axis  
(Relative Rotation).

---

## **FORMAT**

```
#include <Ninja.h>

void njRotateViewXRelative(*v, ang)
NJS_VIEW *v
Angle ang
```

## **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (relative amount of rotation)

## **RETURN VALUE**

None

## **FUNCTION**

- Rotates the view by a relative amount around the X axis.

## EXAMPLE

Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the X axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewXRelative(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

# **njRotateViewXYZ**

Rotates the view around the X, Y, and Z axes  
(Absolute Rotation).

---

## **FORMAT**

```
#include <Ninja.h>

void njRotateViewXYZ(*v, angx, angy, angz)
NJS_VIEW *v
Angle angx
Angle angy
Angle angz
```

## **PARAMETERS**

**v**

view structure pointer

**angx**

rotation angle with respect to X axis (absolute amount of rotation)

**angy**

rotation angle with respect to Y axis (absolute amount of rotation)

**angz**

rotation angle with respect to Z axis (absolute amount of rotation)

## **RETURN VALUE**

None

## **FUNCTION**

- Rotates the view around the X, Y, and Z axes.
- Rotation is applied around the X, Y, and Z axes, in that order.



## EXAMPLE

The following rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (screen rear) by 30 degrees around the X axis, 60 degrees around the Y axis, and 90 degrees around the Z axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njSetView(&v);
njRotateViewXYZ(&v, NJM_DEG_ANG(30), NJM_DEG_ANG(60), NJM_DEG_ANG(90));
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current view is rotated.
- Views (viewpoints) can be moved and rotated by manipulating the view structure in either of two ways: by specifying transformation of view structure matrix in absolute terms, or by manipulating the position (px, py, pz), view direction (vx, vy, vz), and tilt (view roll) of each member in relative terms. This function is of the absolute type.

## RELATED TOPICS

`njSetView()`

`njClearMatrix()`

`NJM_DEG_ANG()`

## **njRotateViewXYZAbsolute**

Rotates the line of view around the X, Y, and Z axes. (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewXYZAbsolutee(*v, angx, angy, angz)
NJS_VIEW *v
Angle angx
Angle angy
Angle angz
```

### **PARAMETERS**

**v**

view structure pointer

**angx**

Rotation angle around X axis (absolute value)

**angy**

Rotation angle around Y axis (absolute value)

**angz**

Rotation angle around Z axis (absolute value)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the line of view around the X, Y, and Z axes.
- Rotation takes place around the X, Y, and Z axes, in that order.

## EXAMPLE

This example rotates the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the X, Y, and Z axes by 30, 60, and 90 degrees, respectively, without changing the view location or direction.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.ay = VIEW_AY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(AVIEW_ROLL);

njRotateViewXYZAbsolute(&v, NJM_DEG_ANG(30), NJM_DEG_ANG(60), NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

[NJM\\_DEG\\_ANG\(\)](#)

## **njRotateViewXYZRelative**

Rotates the view around the X, Y,  
and Z axes (Relative Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewXYZRelative(*v, angx, angy, angz)
NJS_VIEW *v
Angle angx
Angle angy
Angle angz
```

### **PARAMETERS**

**v**

view structure pointer

**angx**

rotation angle around the X axis (relative amount of rotation)

**angy**

rotation angle around the Y axis (relative amount of rotation)

**angz**

rotation angle around the Z axis (relative amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view by a relative amount around the X, Y, and Z axes.
- Rotation is applied around the X, Y, and Z axes, in that order.

## EXAMPLE

The following rotates the view located at (0, 0, 3000) with view orientation (0, 0, -1) (screen rear) by 30 degrees around the X axis, 60 degrees around the Y axis, and 90 degrees around the Z axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewXYZRelative(&v, NJM_DEG_ANG(30), NJM_DEG_ANG(60), NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

[NJM\\_DEG\\_ANG\(\)](#)

## **njRotateViewY**

Rotates the view around the Y axis (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewY(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view around the Y axis.

## EXAMPLE

Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the Y axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njSetView(&v);
njRotateViewY(&v, NJM_DEG_ANG(90));
njClearMatrix();
```

## NOTES

- When parameter `v` is `NULL`, the current view is rotated.
- Views (viewpoints) can be moved and rotated by manipulating the view structure in either of two ways: by specifying transformation of view structure matrix in absolute terms, or by manipulating the position (`px`, `py`, `pz`), view direction (`vx`, `vy`, `vz`), and tilt (view roll) of each member in relative terms. This function is of the absolute type.

## RELATED TOPICS

`njSetView()`

`njClearMatrix()`

`NJM_DEG_ANG()`

## **njRotateViewYAbsolute**

Rotates the view line around the Y axis.  
(Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewYAbsolute(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute value)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the line of view around the Y axis by an absolute amount.



## EXAMPLE

This example rotates the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the Y axis by 90 degrees without changing the view location or direction.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.ay = VIEW_AY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njRotateViewYAbsolute(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)  
[njClearMatrix\(\)](#)

## **njRotateViewYRelative**

Rotates the view around the Y axis  
(Relative Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewYRelative(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (relative amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view by a relative amount around the Y axis.

## EXAMPLE

Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the Y axis.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewYRelative(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

[NJM\\_DEG\\_ANG\(\)](#)

## **njRotateViewZ**

Rotates the view around the Z axis (Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewZ(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view around the Z axis.

## EXAMPLE

Rotates the view located at (0, 0, 3000) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the Z axis.

```
#define VIEW_PX 3000.f
#define VIEW_PY 0.f
#define VIEW_PZ 0.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njSetView(&v);
njRotateViewZ(&v, NJM_DEG_ANG(90));
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current view is rotated.
- Views (viewpoints) can be moved and rotated by manipulating the view structure in either of two ways: by specifying transformation of view structure matrix in absolute terms, or by manipulating the position (px, py, pz), view direction (vx, vy, vz), and tilt (view roll) of each member in relative terms. This function is of the absolute type.
- For details, see the View function View.txt.

## RELATED TOPICS

`njSetView()`

`njClearMatrix()`

`NJM_DEG_ANG()`

## **njRotateViewZAbsolute**

Rotates the view around the Z axis.  
(Absolute Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewZAbsolute(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (absolute value)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view around the Z axis by an absolute amount.

## EXAMPLE

This example rotates the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) around the Z axis by 90 degrees without changing the view location or direction.

```
#define VIEW_APX 3000.f
#define VIEW_APY 0.f
#define VIEW_APZ 0.f
#define VIEW_AVX 0.f
#define VIEW_AY 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.ay = VIEW_AY;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njRotateViewZAbsolute(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, this function acts on the current view.

## RELATED TOPICS

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

[NJM\\_DEG\\_ANG\(\)](#)

## **njRotateViewZRelative**

Rotates the view around the Z axis  
(Relative Rotation).

---

### **FORMAT**

```
#include <Ninja.h>

void njRotateViewZRelative(*v, ang)
NJS_VIEW *v
Angle ang
```

### **PARAMETERS**

**v**

view structure pointer

**ang**

rotation angle (relative amount of rotation)

### **RETURN VALUE**

None

### **FUNCTION**

- Rotates the view by a relative amount around the Z axis.



## EXAMPLE

Rotates the view located at (3000, 0, 0) with orientation (0, 0, -1) (behind the screen) by 90 degrees around the Z axis.

```
#define VIEW_PX 3000.f
#define VIEW_PY 0.f
#define VIEW_PZ 0.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njRotateViewZRelative(&v, NJM_DEG_ANG(90));
njSetView(&v);
njClearMatrix();
```

## NOTES

- When parameter v is NULL, the current matrix is rotated.

## RELATED TOPICS

`njSetView()` `njClearMatrix()` `NJM_DEG_ANG()`

# **njSetAspect**

Sets the screen aspect ratio.

---

## **FORMAT**

```
#include <Ninja.h>

void njSetAspect(ax, ay)
Float ax
Float ay
```

## **PARAMETERS**

**ax**

horizontal screen ratio

**ay**

vertical screen ratio

## **RETURN VALUE**

None

## **FUNCTION**

- Sets the screen aspect ratio.

## **EXAMPLE**

The following sets the vertical:horizontal screen aspect to 2:3.

```
njSetAspect(3.f, 2.f);
```

## **NOTES**

Since the screen aspect is initialized to 1:1 by njInitSystem(), use of this function is optional unless required by the user.

## **RELATED TOPICS**

[njInitSystem\(\)](#)

## **njSetBaseView**

Sets the current view as the base view.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetBaseView(NJS_VIEW *v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**

view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Sets the current view location and viewline orientation as the base view location and view direction.

### EXAMPLE

Moves the view located at (0, 0, 0) and directed toward (0, 0, -1) (toward the inside of the screen) to (50, 50, 50), then sets that view as the base view after reorienting the viewline toward (0, 0, 0).

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 0.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njTranslateViewRelative(50.f, 50.f, 50.f);
njinjLookAtView(&v, 0.f, 0.f, 0.f);
njSetBaseView(&v);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter `v` is `NULL`, this function acts on the current view.

### RELATED TOPICS

`njInitView()`

`njSetView()`

`njClearMatrix()`

## **njSetPerspective**

Sets the perspective in horizontal direction.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetPerspective(ang)
Int ang
```

### **PARAMETERS**

**ang**

perspective in horizontal direction

### **RETURN VALUE**

None

### **FUNCTION**

- Calculates the distance from the specified perspective in horizontal direction to the screen and sets the distance.

### **EXAMPLE**

Sets the perspective in horizontal direction to 120 degrees.

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 );
njSePerspective(120);
```

### **NOTES**

### **RELATED TOPICS**

# njSetScreen

Sets the screen.

## FORMAT

```
#include <Ninja.h>
void njSetScreen(*s)
NJS_SCREEN *s
```

## PARAMETERS

**\*s**

screen structure pointer

## RETURN VALUE

None

## FUNCTION

- Initializes the screen according to screen structure values set by the user.

## EXAMPLE

```
#define SCR_DIST    500.f           //distance from view point (view)
#define SCR_WIDTH   640.f           ///screen width
#define SCR_HEIGHT  480.f           //screen height
#define SCR_CX      SCR_WIDTH/2     //center of screen (X axis)
#define SCR_CY      SCR_HEIGHT/2    //center of screen (Y axis)

NJS_SCREEN scr;

scr.dist = SCR_DIST;
scr.w = SCR_WIDTH;
scr.h = SCR_HEIGHT;
scr.cx = SCR_CX;
scr.cy = SCR_CY;

njSetScreen(&scr);
```

## NOTES

- Since the screen is initialized as indicated above by njInitSystem(), use of this function is optional unless required by the user.

## RELATED TOPICS

[njInitSystem\(\)](#)

## **njSetScreenDist**

Sets the distance from the perspective to the screen.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetScreenDist(dist)
Float dist
```

### **PARAMETERS**

**dist**

distance from the perspective to the screen

### **RETURN VALUE**

None

### **FUNCTION**

- Sets the distance from the perspective to the screen.

### **EXAMPLE**

Sets the distance between the perspective and the screen to 2000.f.

```
njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 );
njSetScreenDist(2000.f);
```

### **NOTES**

### **RELATED TOPICS**

## **njSetView**

Specifies a user-defined view as the current view.

---

### **FORMAT**

```
#include <Ninja.h>

void njSetView(*v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**

view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Specifies a user-defined view as the current view.
- The view matrix is prepared according to position (px, py, pz), view direction (vx, vy, vz), and view tilt (roll) included in the view structure.
- Clears the matrix stack. (njSetView() includes the function of njClearMatrix())

### **EXAMPLE**

```
NJS_VIEW v;

njInitView(&v);
njSetView(&v);
```

### **NOTES**

- The view must be initialized before execution of njSetView().
- For details, see the View function Readme.txt.
- In order to keep the compatibility of programs, njClearMatrix() function still exists. But if njClearMatrix() function is executed, you need to do the same process twice.

### **RELATED TOPICS**

`njInitView()`

`njClearMatrix()`



## njTranslateView

Translates the view along the X, Y, and Z axes  
(Absolute Translation).

---

### FORMAT

```
#include <Ninja.h>

void njTranslateView(*v, x, y, z)
NJS_VIEW *v
Float x
Float y
Float z
```

### PARAMETERS

**v**

view structure pointer

**x**

amount of translation along the X axis (absolute amount of translation)

**y**

amount of translation along the Y axis (absolute amount of translation)

**z**

amount of translation along the Z axis (absolute amount of translation)

### RETURN VALUE

None

### FUNCTION

- Translates the view along the X, Y, and Z axes.

### EXAMPLE

The following translates the view located at (0, 0, 0) with orientation (0, 0, -1) (screen rear) by 100 along the X axis, 200 along the Y axis, and 300 along the Z axis.

```
NJS_VIEW v;  
  
njInitView(&v);  
njSetView(&v);  
njTranslateView(&v, 100.f, 200.f, 300.f);  
njClearMatrix();
```

### NOTES

- The following translates the view located at (0, 0, 0) with orientation (0, 0, -1) (screen rear) by 100 along the X axis, 200 along the Y axis, and 300 along the Z axis.
- Views (viewpoints) can be moved and rotated by manipulating the view structure in either of two ways: by specifying transformation of view structure matrix in absolute terms, or by manipulating the position (px, py, pz), view direction (vx, vy, vz), and tilt (view roll) of each member in relative terms. This function is of the absolute type.

### RELATED TOPICS

`njInitView()`

`njSetView()`

`njClearMatrix()`

## **njTranslateViewAbsolute**

Moves the view location along the X, Y, and Z axes. (Absolute Move).

---

### **FORMAT**

```
#include <Ninja.h>

void njTranslateViewAbsolute(*v, x, y, z)
NJS_VIEW *v
Float x
Float y
Float z
```

### **PARAMETERS**

**v**

view structure pointer

**x**

amount of movement along the X axis (absolute value)

**y**

amount of movement along the Y axis (absolute value)

**z**

amount of movement along the Z axis (absolute value)

### **RETURN VALUE**

None

### **FUNCTION**

Moves the view location along the X, Y, and Z axes by absolute amount. The current view location is taken as the origin.

### EXAMPLE

This example moves the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) along the X, Y, and Z axes by 100, 200, and 300, respectively, without changing the view direction.

```
#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AYV 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

NJS_VIEW v;

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_APZ;
v.avx = VIEW_AVX;
v.avy = VIEW_AYV;
v.avz = VIEW_AVZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);

njTranslateViewAbsolute(&v, 100.f, 200.f, 300.f);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter v is NULL, this function acts on the current view.

### RELATED TOPICS

[njInitView\(\)](#)

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## njTranslateViewRelative

Translates the view along the X, Y, and Z axes (Relative Translation).

---

### FORMAT

```
#include <Ninja.h>

void njTranslateViewRelative(*v, x, y, z)
NJS_VIEW *v
Float x
Float y
Float z
```

### PARAMETERS

**v**

view structure pointer

**x**

amount of translation along X axis (relative amount of translation)

**y**

amount of translation along Y axis (relative amount of translation)

**z**

amount of translation along Z axis (relative amount of translation)

### RETURN VALUE

None

### FUNCTION

- Translates the view relatively along the X, Y, and Z axes.
- The view's current location will be the origin of this translation.

### EXAMPLE

The following translates the view located at (0, 0, 3000) and directed toward (0, 0, -1) (to rear of screen) by 100 in the X direction, 200 in the Y direction, and 300 in the Z direction.

```
#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

NJS_VIEW v;

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);

njTranslateViewRelative(&v, 100.f, 200.f, 300.f);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter v is NULL, the current view is moved.

### RELATED TOPICS

[njInitView\(\)](#)

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njTranslateViewV**

Translates the view along the X, Y, and Z axes  
(Absolute Translation).

---

### **FORMAT**

```
#include <Ninja.h>

void njTranslateViewV( *v, *p)
NJS_VIEW *v
NJS_POINT3 *p
```

### **PARAMETERS**

**v**

view structure pointer

**p**

amount of translation along the X, Y, and Z axes (absolute amount of translation)

### **RETURN VALUE**

None

### **FUNCTION**

- Translates the view along the X, Y, and Z axes.

### **EXAMPLE**

The following translates the view located at (0, 0, 0) with orientation (0, 0, -1) (screen rear) by 100 along the X axis, 200 along the Y axis, and 300 along the Z axis.

```
NJS_VIEW v;
NJS_POINT3 p;

p.x = 100.f;
p.y = 200.f;
p.z = 300.f;

njInitView(&v);
njSetView(&v);
njTranslateView(&v, &p);
njClearMatrix();
```

### NOTES

- When parameter *v* is NULL, the current view is moved.
- Views (viewpoints) can be moved and rotated by manipulating the view structure in either of two ways: by specifying transformation of view structure matrix in absolute terms, or by manipulating the position (*px*, *py*, *pz*), view direction (*vx*, *vy*, *vz*), and tilt (view roll) of each member in relative terms. This function is of the absolute type.

### RELATED TOPICS

`njInitView()`

`njSetView()`

`njClearMatrix()`



## **njTranslateViewVAbsolute**

Moves the view location along the X, Y, and Z axes. (Absolute Move).

---

### **FORMAT**

```
#include <Ninja.h>

void njTranslateViewVAbsolute(*v, *p)
NJS_VIEW *v
NJS_POINT3 *p
```

### **PARAMETERS**

**NJS\_VIEW \*v**

view structure pointer

**NJS\_POINT3 \*p**

amount of movement along each of the X, Y, and Z axes (absolute values)

### **RETURN VALUE**

None

### **FUNCTION**

- Moves the view location along the X, Y, and Z axes by absolute amounts.

### EXAMPLE

This example moves the view located at (0, 0, 3000) and directed toward (0, 0, -1) (into the screen) along the X, Y, and Z axes by 100, 200, and 300, respectively, without changing the view direction.

```
NJS_VIEW v;
NJS_POINT3 p;

#define VIEW_APX 0.f
#define VIEW_APY 0.f
#define VIEW_APZ 3000.f
#define VIEW_AVX 0.f
#define VIEW_AYV 0.f
#define VIEW_AVZ -1.f
#define VIEW_AROLL 0

v.apx = VIEW_APX;
v.apy = VIEW_APY;
v.apz = VIEW_PZ;
v.avx = VIEW_AVX;
v.avy = VIEW_AYV;
v.avz = VIEW_VZ;
v.aroll = NJM_DEG_ANG(VIEW_AROLL);
p.x = 100.f;
p.y = 200.f;
p.z = 300.f;

njTranslateViewVAbsolute(&v, &p);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter v is NULL, this function acts on the current view.

### RELATED TOPICS

[njInitView\(\)](#)

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## njTranslateViewVRelative

Translates the view along the X, Y, and Z axes (Relative Translation).

---

### FORMAT

```
#include <Ninja.h>

void njTranslateViewVRelative(*v, *p)
NJS_VIEW *v
NJS_POINT3 *p
```

### PARAMETERS

**v**

view structure pointer

**p**

amount of translation along the X, Y, and Z axes (relative amount of translation)

### RETURN VALUE

None

### FUNCTION

- Translates the view relatively along the X, Y, and Z axes.
- The view's current location will be the origin of this translation.

### EXAMPLE

The following translates the view located at (0, 0, 3000) and directed toward (0, 0, -1) (to rear of screen) by 100 in the X direction, 200 in the Y direction, and 300 in the Z direction.

```
NJS_VIEW v;
NJS_POINT3 p;

#define VIEW_PX 0.f
#define VIEW_PY 0.f
#define VIEW_PZ 3000.f
#define VIEW_VX 0.f
#define VIEW_VY 0.f
#define VIEW_VZ -1.f
#define VIEW_ROLL 0

v.px = VIEW_PX;
v.py = VIEW_PY;
v.pz = VIEW_PZ;
v.vx = VIEW_VX;
v.vy = VIEW_VY;
v.vz = VIEW_VZ;
v.roll = NJM_DEG_ANG(VIEW_ROLL);
p.x = 100.f;
p.y = 200.f;
p.z = 300.f;

njTranslateViewRelative(&v, &p);
njSetView(&v);
njClearMatrix();
```

### NOTES

- When parameter v is NULL, the current view is moved.

### RELATED TOPICS

[njInitView\(\)](#)

[njSetView\(\)](#)

[njClearMatrix\(\)](#)

## **njUnitBaseViewVector**

Converts the original view vector to a unit vector.

---

### **FORMAT**

```
#include <Ninja.h>

void njUnitCurrentViewVector(*v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**

view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Converts the original view vector specified by the user to a unit vector.

### **EXAMPLE**

```
NJS_VIEW v;

v.apx = v.apy = v.apz = 0.f;
v.avx = v.avy = v.avz = 1.f;
njUnitBaseViewVector(&v);
```

### **NOTES**

- When parameter v is NULL, this function acts on the view matrix of the current view.

```
njUnitBaseViewMatrix(NULL);
```

### **RELATED TOPICS**

## **njUnitCurrentViewVector**

Converts the view vector of the current view to a unit vector.

---

### **FORMAT**

```
#include <Ninja.h>

void njUnitCurrentViewVector(*v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**

view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Converts the current view vector of the view specified by the user to a unit vector.

### **EXAMPLE**

```
NJS_VIEW v;

v.px = v.py = v.pz = 0.f;
v.vx = v.vy = v.vz = 1.f;
njUnitCurrentViewVector(&v);
```

### **NOTES**

- When parameter v is NULL, this function acts on the current view matrix.

```
njUnitCurrentViewVector(NULL);
```

### **RELATED TOPICS**

## **njUnitViewMatrix**

Sets a unit matrix to the view matrix.

---

### **FORMAT**

```
#include <Ninja.h>

void njUnitViewMatrix(*v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**

view structure pointer

### **RETURN VALUE**

None

### **FUNCTION**

- Sets a unit matrix to the view matrix of the view specified by the user.

### **EXAMPLE**

```
NJS_VIEW v;
njUnitViewMatrix(&v);
```

### **NOTES**

- When parameter v is NULL, the unit matrix is set to the view matrix of the current view.

```
njUnitViewMatrix(NULL);
```

### **RELATED TOPICS**

## **njUnitViewVector**

Converts the view vector to a unit vector.

---

### **FORMAT**

```
#include <Ninja.h>

void njUnitViewVector(*v)
NJS_VIEW *v
```

### **PARAMETERS**

**v**

pointer to the view structure

### **RETURN VALUE**

None

### **FUNCTION**

- Converts both the view vector of the current user-specified view and the base view vector to unit vectors.

### **EXAMPLE**

```
NJS_VIEW v;

v.px = v.py = v.pz = 0.f;
v.apx = v.apy = v.apz = 0.f;
v.vx = v.vy = v.vz = 1.f;
v.avx = v.avy = v.avz = 1.f;
njUnitViewVector(&v);
```

### **NOTES**

- When parameter v is NULL, the operation is performed on the view matrix of the current view.

```
njUnitViewMatrix(NULL);
```

### **RELATED TOPICS**





# 11. Texture Functions

---

## Contents

njCalcTexture .....	Calculates the remaining texture memory size. ....	NLS-344
njGetTextureNumG.....	Obtains the global index number of the current texture. ....	NLS-346
njInitTexture .....	Sets the area used for storing texture information. ....	NLS-347
njLoadCacheTexture.....	Sets the cache information area. ....	NLS-349
njLoadCacheTextureNum .....	Loads a texture by texture number. ....	NLS-351
njLoadCacheTextureNumG .....	Loads texture number globalIndex from cache memory into texture memory. ....	NLS-353
njLoadTexture .....	Loads a texture. ....	NLS-355
njLoadTextureNum .....	Loads textures. ....	NLS-358
njReleaseCacheTextureNum ....	Release cache memory. ....	NLS-360
njReleaseCacheTextureAll .....	Release all cache memory. ....	NLS-362
njReleaseCacheTextureNumG .	Releases cache memory. ....	NLS-364
njReleaseTexture .....	Releases texture memory. ....	NLS-366
njReleaseTextureNum.....	Releases texture memory. ....	NLS-368
njReleaseTextureNumG .....	Releases texture memory. ....	NLS-370
njReleaseTextureAll .....	Release all texture memory. ....	NLS-372
njSetTexture .....	Set current texture list. ....	NLS-374
njSetTextureInfo .....	Set data into texture name structure. ....	NLS-376
njSetTextureName .....	Set data into texture name structure. ....	NLS-379
njSetTextureNum.....	Sets a texture number as the current texture. ....	NLS-381
njSetTextureNumG .....	Sets the current texture to a global index number. ....	NLS-384
njReLoadTextureNum.....	Reloads Textures. ....	NLS-386
njReLoadTextureNumG .....	Reloads Textures. ....	NLS-388
njRenderTextureNum .....	Renders Texture Area. ....	NLS-390
njRenderTextureNumG .....	Renders at the Texture Area. ....	NLS-392
njSetRenderWidth .....	Sets Stride Value. ....	NLS-393

## **njCalcTexture**

Calculates the remaining texture memory size.

---

### **FORMAT**

```
#include <Ninja.h>
Uint32 njCalcTexture(flag);
Uint32 flag
```

### **PARAMETERS**

**NJD\_TEXMEM\_FREESIZE**

free size in total

**NJD\_TEXMEM\_MAXBLOCK**

max free size in block

**NJD\_TEXMEM\_MAXSIZE**

max size in texture memory

### **RETURN VALUE**

Amount of remaining texture memory

### **FUNCTION**

- Calculates total free size or max free size in block of texture memory.

## EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};
Uint32 calc;

/* Prepare to read two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);
njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
                 NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
                 NJD_TEXATTR_GLOBALINDEX);

/* Load textures */
njLoadTexture(&texlist);

/* Calculates the free size in total */
calc = njCalcTexture(NJD_TEXMEM_FREESIZE);

/* Output on screen */
njPrintD(NJM_LOCATION(10, 10), calc, 10);
```

## NOTES

- `njInitTexture()` must be executed before calling this function.

## RELATED TOPICS

## **njGetTextureNumG**      Obtains the global index number of the current texture.

---

### **FORMAT**

```
#include <Ninja.h>
Uint32 njGetTextureNumG(void);
```

### **PARAMETERS**

None

### **RETURN VALUE**

#### **Successful**

Global index number in the range 0 to 0xFFFFFFFF0

#### **Failed**

0xFFFFFFFF

### **FUNCTION**

- Obtains the global index number of the current texture.

### **EXAMPLE**

```
Uint32 globalIndex;

/* Get the global index number for the current texture previously set */
globalIndex = njGetTextureNumG();

/* Set the global index #0 to the current texture. */
njSetTextureNumG(0);

                                :
Drawing textures...           :
                                :

/* Put the current texture back */
njSetTextureNumG(globalIndex);
```

### **NOTES**

- Current texture must be set previously by njSetTextureNum, njSetTextureNumG.

### **RELATED TOPICS**

[njSetTextureNum](#) [njSetTextureNumG](#)

## **njInitTexture**                      Sets the area used for storing texture information.

---

### **FORMAT**

```
#include <Ninja.h>
void njInitTexture(*addr, n);
NJS_TEXMEMLIST *addr
Uint32 n
```

### **PARAMETERS**

**\*addr**

pointer to the area holding n NJS\_TEXMEMLIST structures

**n**

number of textures to be used

### **RETURN VALUE**

None

### **FUNCTION**

- Establishes the area for storing textures by setting addr with the pointer to the NJS structure area for holding the n textures to be used.
- This function must be executed before loading textures.

### EXAMPLE

```
/* Set two textures */
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist={texname,2};

/* Prepare to read two textures*/
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist,2);
/* Load the textures */
njSetTextureName(&texname[0],"file1.pvr",0,NJD_TEXATTR_TYPE_FILE|
NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1],"file2.pvr",1,NJD_TEXATTR_TYPE_FILE|
NJD_TEXATTR_GLOBALINDEX);

njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);
/* Set current texture to the texlist #0 */
njSetTextureNum(0);
```

### NOTES

- The area set by this function is used internally by texture-related functions.
- For details, see the texture document.

### RELATED TOPICS

[njLoadTexture](#)

[njLoadTextureNum](#)

[njLoadTextureNumG](#)

## **njLoadCacheTexture**

Sets the cache information area.

---

### **FORMAT**

```
#include <Ninja.h>
Sint32      njLoadCacheTexture(*texlist);
NJS_TEXLIST *texlist
```

### **PARAMETERS**

**\*texlist**

pointer to the NJS\_TEXLIST structure

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Load textures set to the texlist structure from cache memory to texture memory.

### EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read two textures in */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

/* Specify to read file into cache */
njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
/* Specify to read file into cache */
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);

/* Load texture into cache */
njLoadTexture(&texlist);

/* Set texlist as the current texture */
njSetTexture(&texlist);

/* Read texture from cache into texture memory*/
njLoadCacheTexture(&texlist);

/* Set current texture as texlist #0    file1.pvr*/
njSetTextureNum(0);

Drawing textures
```

### NOTES

- The current texture list must be set by njSetTexture.
- The specified texture must be loaded into cache memory.
- See also Texture guide for more reference.

### RELATED TOPICS

[njLoadCacheTextureNumG](#)



## **njLoadCacheTextureNum**

Loads a texture by texture number.

---

### **FORMAT**

```
#include <Ninja.h>
 Sint32 njLoadCacheTextureNum(n);
 Uint32 n
```

### **PARAMETERS**

**n**

texture number in current texture list

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Loads a texture by texture number

### EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX )
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);

/* Load texture and read into cache */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Read textures from cache into texture memory*/
njLoadCacheTextureNum(0);
njLoadCacheTextureNum(1);

/* Set current texture as texture #0 in texlist file1.pvr*/
njSetTextureNum(0);

:

Drawing texture with texture in file1.pvr

:

/* Release texture #0 in cache */
njReleaseCacheTextureNum(0);
```

### NOTES

- Current texture list must be specified by njSetTexture.
- The specified texture must be loaded into cache memory.
- Even if texture in cache memory is released, the one in texture memory will not be released. Refer to Texture Guide for more details.

### RELATED TOPICS

[njLoadCacheTexture](#) [njLoadCacheTextureNumG](#)

## **njLoadCacheTextureNumG**

Loads texture number `globalIndex` from cache memory into texture memory.

---

### **FORMAT**

```
#include <Ninja.h>

Sint32  njLoadCacheTextureNumG( globalIndex );

Uint32  globalIndex
```

### **PARAMETERS**

**globalIndex**

Global index number

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Loads texture number `globalIndex` from from cache memory into texture memory.

### EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 100, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 200, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);

/* Load texture and read into cache */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Read textures from cache into texture memory*/
njLoadCacheTextureNumG(100); /* file1.pvr */
njLoadCacheTextureNumG(200); /* file2.pvr */

/* Set current texture as texture #0 in texlist file1.pvr*/

njSetTextureNumG(100);

:

Drawing texture with texture in file1.pvr

:
```

### NOTES

- The specified texture must be loaded into cache memory.
- Even if texture in cache memory is released, the one in texture memory will not be released. Refer to Texture Guide for more details.

### RELATED TOPICS

[njLoadCacheTexture](#)

[njLoadCacheTextureNum](#)

## njLoadTexture

Loads a texture.

---

### FORMAT

```
#include <Ninja.h>
 Sint32 njLoadTexture(*texlist);
NJS_TEXLIST *texlist
```

### PARAMETERS

**\*texlist**

pointer to the NJS\_TEXLIST structure

### RETURN VALUE

**Successful**

1

**Failed**

-1

### FUNCTION

- Loads the texture set into the texlist structure into texture memory or cache memory.

#### Notes on texture creation

1. Set texture name structure for textures.

#### NJS\_TEXNAME

```
void      *filename
Uint16     attr
Uint32     texaddr
```

**\*filename**

Filename when it is a PVR format texture file.

The NJS\_TEXINFO structure pointer when loaded from memory.

#### globalIndex

The texture global number defined within the application.

It will be treated as the same texture if the global index number is the same, even if the texture list is different.

Valid value: 0 thru 0xFFFFFFFF. 0xFFFFFFFF is prohibited to use.

### **attr**

Specifies the origin and the destination of loading textures, and performs OR for each tag.

### **[the origin of load]**

#### **NJD\_TEXATTR\_TYPE\_FILE**

Loads the PVR format files. Specifies filename with \*filename.

#### **NJD\_TEXATTR\_TYPE\_MEMORY**

Loads from memory. Sets NJS\_TEXINFO structure pointer with \*filename.

Refer to following for setting NJS\_TEXINFO.

### **[the destination](loads into texture memory if not specified)**

#### **NJD\_TEXATTR\_CACHE**

Loads into cache memory only.

#### **NJD\_TEXATTR\_BOTH**

Loads into both texture memory and cache memory.

### **[global index specification]**

#### **NJD\_TEXATTR\_GLOBALINDEX**

This is used if global index within the file is not used.

#### **texaddr**

Specifies global index at initialization if NJD\_TEXATTR\_GLOBALINDEX is specified with attr.

After loading textures, it holds the address of NJS\_TEXMEMLIST.

*2.If textures are loaded from memory, it is necessary to set NJS\_TEXINFO structure.*

### **NJS\_TEXINFO structure**

```
void*          texaddr;  
NJS_TEXSURFACE texsurface;
```

#### **texaddr**

Stores address when cache texture is specified.

#### **texsurface**

Sets data after loading textures.

### **NJS\_TEXLIST structure**

```
NJS_TEXNAME      *textures;  
Uint32           nbTexture;
```

### **textures**

Sets the pointer for NJS\_TEXNAME structure that has texture information

nbTexture

number of textures

### **EXAMPLE**

```
NJS_TEXNAME texname[2];  
  
NJS_TEXLIST texlist = {texname, 2};  
  
/* Prepare to read in two textures */  
NJS_TEXMEMLIST texmemlist[2];  
  
njInitTexture(texmemlist, 2);  
  
/* Set the two textures */  
njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |  
                                                         NJD_TEXATTR_GLOBALINDEX);  
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |  
                                                         NJD_TEXATTR_GLOBALINDEX);  
  
/* load texture */  
njLoadTexture(&texlist);  
  
/* Set texlist as the current texture list */  
njSetTexture(&texlist);  
  
/* Set current texture to texlist #0 */  
njSetTextureNum(0);
```

## **NOTES**

- njInitTexture must be executed prior to executing this function.
- Refer to Texture Guide for more details.

## **RELATED TOPICS**

[njInitTexture](#)

[njLoadTextureNum](#)

# **njLoadTextureNum**

Loads textures.

## **FORMAT**

```
#include <Ninja.h>
Sint32 njLoadTextureNum(n);
Uint32 n
```

## **PARAMETERS**

**n**

texture number in the current texture list

## **RETURN VALUE**

**Successful**

1

**Failed**

-1

## **FUNCTION**

- Load the number n texture in the current texture list into either texture memory or cache memory.



## EXAMPLE

```
/* Set two textures */
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read textures for two */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

/* Set the two textures */
njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
                                                         NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
                                                         NJD_TEXATTR_GLOBALINDEX);

/* Set texlist as the current texture list */
/* It must be done first */
njSetTexture(&texlist);

/* Load texture #0 file1.pvr */
njLoadTextureNum(0);

/* Load texture #1 file2.pvr */
njLoadTextureNum(1);

/* Set current texture as texture #0 in the texlist */
njSetTextureNum(0);
```

## NOTES

- njInitTexture and njSetTexture must be executed prior to executing this function. Refer to Texture Guide for more details.

## RELATED TOPICS

[njInitTexture](#) [njLoadTexture](#)

## **njReleaseCacheTextureNum**

Release cache memory.

---

### **FORMAT**

```
#include <Ninja.h>
Sint32 njReleaseCacheTextureNum(n);
Uint32 n
```

### **PARAMETERS**

**n**

Texture number in the current texture list

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Release the cache memory for texture n

## EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
/* Load texture and read into cache */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Read texture from cache into texture memory*/
njLoadCacheTextureNum(0);
njLoadCacheTextureNum(1);

/* Set current texture to texlist #0 file1.pvr*/
njSetTextureNum(0);

:

Drawing texture with texture in file1.pvr

:

/* Release texture #0 in cache */
njReleaseCacheTextureNum(0);
```

## NOTES

- Current texture list must be specified by njSetTexture.
- The specified texture must be loaded into cache memory.
- Even if texture in cache memory is released, the one in texture memory will not be released. Refer to Texture Guide for more details.

## RELATED TOPICS

[njReleaseCacheTextureAll](#)

[njReleaseCacheTextureNumG](#)

# **njReleaseCacheTextureAll**

Release all cache memory.

---

## **FORMAT**

```
#include <Ninja.h>
void njReleaseCacheTextureAll(void);
```

## **PARAMETERS**

None

## **RETURN VALUE**

None

## **FUNCTION**

- Release all cache memory

## EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
/* Load texture and read into cache */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Read texture from cache into texture memory*/
njLoadCacheTextureNum(0);
njLoadCacheTextureNum(1);

/* Set current texture to texlist #0 file1.pvr*/
njSetTextureNum(0);

                                                                    :
Drawing texture with texture in file1.pvr                               :
                                                                    :

/* Release all textures in cache */
njReleaseCacheTextureAll();
```

## NOTES

Even if texture in cache memory is released, the one in texture memory will not be released. Refer to Texture Guide for more details.

## RELATED TOPICS

## **njReleaseCacheTextureNumG**

Releases cache memory.

### **FORMAT**

```
#include <Ninja.h>
Sint32 njReleaseCacheTextureNumG(globalIndex);
Uint32 globalIndex
```

### **PARAMETERS**

**globalIndex**

global index number

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Releases cache memory for globalIndex

## EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 100, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 200, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_TYPE_CACHE | NJD_TEXATTR_GLOBALINDEX);

/* Load texture and read into cache */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Read textures from cache into texture memory*/
njLoadCacheTextureNum(0);
njLoadCacheTextureNum(1);

/* Set current texture as texture #0 in texlist file1.pvr*/
njSetTextureNum(0);

:
Drawing texture with texture in file1.pvr
:

/* Release the global index #100 texture stored in cache */
njReleaseCacheTextureNumG(100);
```

## NOTES

- The specified texture must be loaded into cache memory.
- Even if texture in cache memory is released, the one in texture memory will not be released. Refer to Texture Guide for more details.

## RELATED TOPICS

# **njReleaseTexture**

Releases texture memory.

---

## **FORMAT**

```
#include <Ninja.h>
Sint32 njReleaseTexture(*texlist);
NJS_TEXLIST *texlist
```

## **PARAMETERS**

**\*texlist**

NJS\_TEXLIST structure pointer

## **RETURN VALUE**

**Successful**

1

**Failed**

-1

## **FUNCTION**

- Releases texture memory of texlist.



## EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXNAME texname2[2];

NJS_TEXLIST texlist = {texname, 2};
NJS_TEXLIST texlist2 = {texname2, 2};

/* Prepare to read four textures */
NJS_TEXMEMLIST texmemlist[4];

njInitTexture(texmemlist, 4);

njSetTextureName(&texname[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname[1], "file2.pvr", 1, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname2[0], "file1.pvr", 0, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
njSetTextureName(&texname2[1], "file3.pvr", 2, NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);

/* load texture */
njLoadTexture(&texlist);
njLoadTexture(&texlist2);

/* Release the texture in texlist */
njReleaseTexture(&texlist);

/*
    Even if texlist is released, since file1.pvr is in texlist2
also,
    file.pvr will not be released from texture memory.
    ** However file1.pvr cannot be read with texlist unless it is
loaded
    again.
*/
```

## NOTES

- As noted above, if the texture is loaded in another texture list, the texture will not be released from texture memory unless all other occupying lists release the texture.
- Refer to Texture Guide for more details.

## RELATED TOPICS

## **njReleaseTextureNum**

Releases texture memory.

---

### **FORMAT**

```
#include <Ninja.h>
 Sint32 njReleaseTextureNum(n);
 Uint32 n
```

### **PARAMETERS**

**n**

number of texture

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Releases n texture in the current texture list from texture memory.

## EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 100,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname[1], "file2.pvr", 200,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

/* load texture */
njLoadTexture(&texlist);

/* set current texture list*/
njSetTexture(&texlist);

/* Release texture #0 in texlist   file.pvr*/
njReleaseTextureNum(0);
```

## NOTES

- If the texture is registered in another texture list, the texture will not be released from texture memory unless all other occupying lists release the texture.
- Refer to Texture Guide for more details.

## RELATED TOPICS

`njReleaseTextureAll`  
`njReleaseTexture`  
`njReleaseTextureNumG`

# **njReleaseTextureNumG**

Releases texture memory.

---

## **FORMAT**

```
#include <Ninja.h>
Sint32 njReleaseTextureNumG(globalIndex);
Uint32 globalIndex
```

## **PARAMETERS**

**globalIndex**

Global index number

## **RETURN VALUE**

**Successful**

1

**Failed**

-1

## **FUNCTION**

- Releases texture specified by globalIndex in the current texture list from texture memory.

## EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 100,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname[1], "file2.pvr", 200,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

/* load texture */
njLoadTexture(&texlist);

/* set current texture list*/
njSetTexture(&texlist);

/* Release the global index number 100 texture
 * in the texlist file1.pvr
 */
njReleaseTextureNumG(100);
```

## NOTES

- If the texture is registered in another texture list, the texture will not be released from texture memory unless all other occupying lists release the texture.
- Refer to Texture Guide for more details.

## RELATED TOPICS

[njReleaseTextureAll](#)

[njReleaseTexture](#)

[njReleaseTextureNum](#)

# **njReleaseTextureAll**

Release all texture memory.

---

## **FORMAT**

```
#include <Ninja.h>
void njReleaseTextureAll(void);
```

## **PARAMETERS**

None

## **RETURN VALUE**

None

## **FUNCTION**

- Release all texture memory.

## EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read in two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 0,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname[1], "file2.pvr", 1,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

/* Load texture */
njLoadTexture(&texlist);

/* Set the current texture list */
njSetTexture(&texlist);

/* set current texture to texture #0 in texlist file1.pvr */
njSetTextureNum(0);

:

Drawing textures

:

/* release all textures */
njReleaseTextureAll();
```

## NOTES

- It must be loaded again if the texture is needed.
- See also Texture Guide for more details.

## RELATED TOPICS

[njReleaseTexture](#)

[njReleaseTextureNum](#)

[njReleaseTextureNumG](#)

## njSetTexture

Set current texture list.

### FORMAT

```
#include <Ninja.h>
 Sint32 njSetTexture(*texlist);
 NJS_TEXLIST *texlist
```

### PARAMETERS

**\*texlist**

NJS\_TEXLIST structure pointer

### RETURN VALUE

**Successful**

1

**Failed**

-1

### FUNCTION

- Set texlist as the curenent texture list

### EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXNAME texname2[2];

NJS_TEXLIST texlist = {texname, 2};
NJS_TEXLIST texlist = {texname2, 2};

/* Prepare to read three textures */
NJS_TEXMEMLIST texmemlist[3];

njInitTexture(texmemlist, 3);

njSetTextureName(&texname[0], "file1.pvr", 0,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname[1], "file2.pvr", 1,
```



```

NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname2[0], "file1.pvr", 0,

NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname2[1], "file3.pvr", 1,

NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

/* Load texture */
njLoadTexture(&texlist);

/* Load texture */
njLoadTexture(&texlist2);

/* Set current texture list to texlist */
/* no problem if it's done prior to executing njLoadTexture */
njSetTexture(&texlist);

/* Set current texture as #0 texture in texlist file1.pvr*/
njSetTextureNum(0);

:

Drawing textures using file1.pvr

:

/* Set texlist2 as the current texture list */
njSetTexture(&texlist2);

/* Set current texture to texlist2 #1 texture file3.pvr */
njSetTextureNum(1);

:

Drawing textures using file3.pvr

:
```

## NOTES

- Current texture list won't be changed until next time njSetTexture is executed.
- All texture related functions such as njXXXXNum and njXXXXNumG will work with current texture list.
- Refer to Texture guide for more information.

## RELATED TOPICS

[njSetTextureNum](#)

[njSetTextureNumG](#)

## **njSetTextureInfo**

Set data into texture name structure.

---

### **FORMAT**

```
#include <Ninja.h>
njSetTextureInfo(*info, *tex, Type, nWidth, nHeight)
NJS_TEXINFO *info
Uint16      *tex
Sint32      Type
Sint32      nWidth
Sint32      nHeight
```

### **PARAMETERS**

**\*info**

texture information (output)

**\*tex**

memory texture pointer

**Type**

texture type

**nWidth**

texture width size

**nHeight**

texture height size

### **RETURN VALUE**

None

### **FUNCTION**

- Sets texture data into texture information structure.
- For Type, color format and category code are set. Set all the information into addr of njSetTextureName.

**Color format**

NJD_TEXFMT_ARGB_1555	
NJD_TEXFMT_RGB_565	
NJD_TEXFMT_ARGB_4444	
NJD_TEXFMT_YUV_422	currently not used
NJD_TEXFMT_BUMP	currently not used

**Category code**

NJD_TEXFMT_TWIDDLED	
NJD_TEXFMT_TWIDDLED_MM	
NJD_TEXFMT_VQ	currently not used
NJD_TEXFMT_VQ_MM	currently not used
NJD_TEXFMT_PALETTIZE4	currently not used
NJD_TEXFMT_PALETTIZE4_MM	currently not used
NJD_TEXFMT_PALETTIZE8	currently not used
NJD_TEXFMT_PALETTIZE8_MM	currently not used
NJD_TEXFMT_RECTANGLE	
NJD_TEXFMT_STRIDE	

### EXAMPLE

```
NJS_TEXINFO Info;
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureInfo(&Info, Image,
NJD_TEXFMT_RECTANGLE |
NJD_TEXFMT_ARGB_1555
, 256, 256);

njSetTextureName(&texname[0], "file0.pvr", 0,
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname[1], &Info, 1,
NJD_TEXATTR_TYPE_MEMORY |
NJD_TEXATTR_GLOBALINDEX );

/* Load texture */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Set the texlist #1 texture as the current texture Image */
njSetTextureNum(1);

:
Drawing textures using memorytexture Image.
:
```

### NOTES

- See Texture Guide for more information

### RELATED TOPICS

[njSetTextureName](#)

## **njSetTextureName**

Set data into texture name structure.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetTextureName(*texname, *addr, globalIndex, attr)
NJS_TEXNAME *texname
void *addr
Uint32 globalIndex
Uint32 attr
```

### **PARAMETERS**

**\*texname**

NJS\_TEXNAME structure pointer

**\*addr**

file name or NJS\_TEXINFO structure pointer

**globalIndex**

global index

**attr**

texture attribute

### **RETURN VALUE**

None

### **FUNCTION**

Set filename to addr when loading textures from a file. NJD\_TEXATTR\_TYPE\_FILE should be specified to attr.

With PVR format textures, if global index is not used for the textures, set NJD\_TEXATTR\_GLOBALINDEX to attr and global index to globalIndex.

For memory texture, set NJS\_TEXINFO structure pointer that is set by njSetTextureInfo to addr.

Also, set NJD\_TEXATTR\_TYPE\_MEMORY to attr, and global index to globalIndex.

### EXAMPLE

```
NJS_TEXNAME texname[2];

NJS_TEXLIST texlist = {texname, 2};

/* Prepare to read two textures */
NJS_TEXMEMLIST texmemlist[2];

njInitTexture(texmemlist, 2);

njSetTextureName(&texname[0], "file1.pvr", 0,
                 NJD_TEXATTR_TYPE_FILE |
                 NJD_TEXATTR_GLOBALINDEX );

njSetTextureName(&texname[1], "file2.pvr", 1,
                 NJD_TEXATTR_TYPE_FILE |
                 NJD_TEXATTR_GLOBALINDEX );

/* Load texture */
njLoadTexture(&texlist);

/* Set texlist as the current texture list */
njSetTexture(&texlist);

/* Set texture 0 from texlist as current texture file1.pvr */
njSetTextureNum(0);

:
/* Omitted texture drawing Drawing with texture file1.pvr */
:
```

### NOTES

- The global index set here is once stored to texaddr in NJS\_TEXNAME structure and passed to NJS\_MEMLIST structure globalIndex when loading textures.
- At that time texaddr is overwritten.
- So you'll need to set it again if you load textures, release them and reload them.
- Refer to Texture Guide for more information.

### RELATED TOPICS

[njSetTextureInfo](#)

## **njSetTextureNum**

Sets a texture number as the current texture.

---

### **FORMAT**

```
#include <Ninja.h>
 Sint32 njSetTextureNum(n);
 Uint32 n
```

### **PARAMETERS**

**n**

texture number

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

Sets texture number *n* from the current texture list as the current texture.

Subsequently, this remains the current texture until execution of `njSetTextureNum` or `njSetTextureNumG`.

#### **About the texture number of the current texture**

As shown in the example below, when textures are created, texture numbers are sequentially allocated in the series 0, 1, 2, 3,... from the beginning of a `NJS_TEXNAME` structure which is set in the `NJS_TEXLIST` structure.

```
NJS_TEXNAME texname[5];
NJS_TEXLIST texlist={texname,5};
```

```
/* Texture number 0 */
njSetTextureName(&texname[0], "file1.pvr", 10,
```

```
    NJD_TEXATTR_TYPE_FILE |
    NJD_TEXATTR_GLOBALINDEX);
```

```
/* Texture number 1 */
njSetTextureName(&texname[1], "file2.pvr", 11,
```

```
/* Texture number 2 */
njSetTextureName(&texname[2], "file1.pvr", 12,
```

```
/* Texture number 3 */
njSetTextureName(&texname[3], "file2.pvr", 13,
```

```
/* Texture number 4 */
njSetTextureName(&texname[4], "file1.pvr", 14,
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX);
```

## EXAMPLE

```
NJS_TEXNAME texname[2];
NJS_TEXNAME texname2[2];
```

```
NJS_TEXLIST texlist = {texname, 2};
NJS_TEXLIST texlist = {texname2, 2};
```

```
/* Prepare to load a total of 4 textures */
NJS_TEXMEMLIST texmemlist[4];
```

```
njInitTexture(texmemlist, 4);
```

```
/* Load texture */
njLoadTexture(&texlist);
```

```
njSetTextureName(&texname[0], "file1.pvr", 0,
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );
```

```
njSetTextureName(&texname[1], "file2.pvr", 1,
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );
```

```
njSetTextureName(&texname2[0], "file3.pvr", 2,
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );
```

```
njSetTextureName(&texname2[1], "file4.pvr", 3,
```

```
NJD_TEXATTR_TYPE_FILE |
NJD_TEXATTR_GLOBALINDEX );
```



```
/* Load texture */
njLoadTexture(&texlist2);

/* Set current texture list as texlist */
njSetTexture(&texlist);

/* Set texture 0 from texlist as current texture file1.pvr */
njSetTextureNum(0);

:

/* Omitted texture drawing Drawing with texture file1.pvr */
:

/* Set texlist2 as current texture list */
njSetTexture(&texlist2);

/* Set texture 1 from texlist2 as current texture file4.pvr */
njSetTextureNum(1);

:

/* Omitted texture drawing Drawing with texture file1.pvr */
:
```

## NOTES

- The specified texture must be present in texture memory.
- For details, see the texture document.

## RELATED TOPICS

[njSetTexture](#)

[njSetTextureNumG](#)

## **njSetTextureNumG**

Sets the current texture to a global index number.

---

### **FORMAT**

```
#include <Ninja.h>
Sint32 njSetTextureNumG(globalIndex);
Uint32 globalIndex
```

### **PARAMETERS**

**globalIndex**

global index number

### **RETURN VALUE**

**Successful**

1

**Failed**

-1

### **FUNCTION**

- Sets the texture identified by global index number `globalIndex` as the current texture. Subsequently, this remains the current texture until execution of `njSetTextureNum` or `njSetTextureNumG`.

### **EXAMPLE**

```
NJS_TEXNAME texname[2];
NJS_TEXNAME texname2[2];

NJS_TEXLIST texlist = {texname, 2};
NJS_TEXLIST texlist = {texname2, 2};

/* Prepare to read in textures for four */
NJS_TEXMEMLIST texmemlist[4];

njInitTexture(texmemlist, 4);

njSetTextureName(&texname[0],
                                     "file1.pvr", 100,
                                     NJD_TEXATTR_TYPE_FILE |
                                     NJD_TEXATTR_GLOBALINDEX );
```

```
njSetTextureName(&texname[1],  
                "file2.pvr",200,  
                NJD_TEXATTR_TYPE_FILE |  
                NJD_TEXATTR_GLOBALINDEX );  
  
njSetTextureName(&texname2[0],  
                "file3.pvr",300,  
                NJD_TEXATTR_TYPE_FILE |  
                NJD_TEXATTR_GLOBALINDEX );  
  
njSetTextureName(&texname2[1],  
                "file4.pvr",400,  
                NJD_TEXATTR_TYPE_FILE |  
                NJD_TEXATTR_GLOBALINDEX );  
  
/* Load texture */  
njLoadTexture(&texlist);  
  
/* Load texture */  
njLoadTexture(&texlist2);  
  
/* Set global index #100 texture  
* as the current texture file1.pvr  
*/  
njSetTextureNumG(100);  
  
: Drawing textures (using file1.pvr textures.)  
:  
  
/* Set the 3rd texture in global index  
* as the current texture file4.pvr  
*/  
njSetTextureNumG(400);  
  
:  
  
Drawing textures (using file4.pvr textures.)  
:
```

## NOTES

- Specified texture must be located in texture memory.
- Refer to Texture Guide for more details.

## RELATED TOPICS

[njSetTexture](#)

[njSetTextureNum](#)

# njReloadTextureNum

Reloads Textures.

## FORMAT

```
#include <Ninja.h>
Sint32 njReloadTextureNum(n, *texaddr, attr, lod);
Uint32 n
void *texaddr
Uint32 attr
Uint32 lod
```

## PARAMETERS

**n**

texture number of the current texture list

**\*texaddr**

address for either a filename or a texture memory

**attr**

texture attribute

**lod**

mipmap level

## RETURN VALUE

**Success**

1

**Failure**

-1

## ERROR VALUE

None

## FUNCTION

- Reloads texture to the texture number *n* of the current texture list.
- Textures that can be reloaded must be the same format and size as previously loaded.
- The reloading texture overwrites the previous texture.
- When loading textures from a file, specify `NJD_TEXATTR_TYPE_FILE` for `attr`. When loading textures from a memory, specify `NJD_TEXATTR_TYPE_MEMORY` for `attr`.
- Also, for mipmap textures, specifying `lod` allows reloading corresponding mipmap level. For example, setting 128 to `lod` will reload only 128x128 level mipmap textures.
- Specifying 0 to `lod` will reload all levels of mipmap textures.
- When loading from memory, it reloads from the address specified by `texaddr` to the level specified by `lod`.

## EXAMPLE

### Suppose

- 256x256 mipmap is loaded as texture #0 in the current texture list,
- 128x128 mipmap is loaded as texture #1 in the current texture list,
- 256x128 rectangle texture is loaded as texture #2 in the current texture list.

```
/*
                                Change texture #0 texture, mipmap level 128x128 to file1.pvr.
                                (Here, file1.pvr is 256x256 mipmap texture.)
*/
njReloadTextureNum(0, "file1.pvr", NJD_TEXATTR_TYPE_FILE, 128);

/*
                                Reloads all the textures of texture #1.
                                (Here, file2.pvr is 128x128 mipmap texture.)
*/
njReloadTextureNum(1, "file2.pvr", NJD_TEXATTR_TYPE_FILE, 0);

/*
                                Reloads all the textures of texture #2.
                                (Here file3.pvr is 256x128 rectangle texture.)
*/
njReloadTextureNum(2, "file3.pvr", NJD_TEXATTR_TYPE_FILE, 0);
```

## NOTES

- For memory textures, it specifies the beginning of texture specified by `lod`.
- Please refer to texture documentation for more details.

## RELATED TOPICS

`njLoadTextureNum`

`njLoadTextureNumG`

`njReloadTextureNumG`

# njReloadTextureNumG

Reloads Textures.

---

## FORMAT

```
#include <Ninja.h>
Sint32 njReloadTextureNumG(globalIndex,*texaddr,attr,lod);
Uint32 globalIndex
void *texaddr
Uint32 attr
Uint32 lod
```

## PARAMETERS

**globalIndex**

global index number

**\*texaddr**

address of either a file name or a texture memory

**attr**

texture attribute

**lod**

mipmap level

## RETURN VALUE

**Successful**

1

**Failed**

-1

## ERROR VALUE

None

## FUNCTION

- Reloads the texture specified by globalIndex.

## EXAMPLE

## NOTES

## RELATED TOPICS

`njLoadTextureNum`

`njLoadTextureNumG`

`njReloadTextureNum`

# **njRenderTextureNum**

Renders Texture Area.

## **FORMAT**

```
#include <Ninja.h>
void njRenderTextureNum(n);
Uint32 n
```

## **PARAMETERS**

**n**

texture number in the current texture list

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Renders to the number n texture of the current texture list.
- Textures that can render are either NJD\_TEXFMT\_RECTANGLE or NJD\_TEXFMT\_STRIDE.
- If texture area is smaller than the screen, it adjusts the upper left and performs rendering.
- Stride value must be set by njSetRenderWidth if NJD\_TEXFMT\_STRIDE
- Normally, the stride value is the same as rendering area width.

## **EXAMPLE**

```
void njUserInit(void)
{

    color mode                                /* When render texture is used, set njInitSystem frame buffer
                                           with the texture color mode used for render texture.
                                           */
    njInitSystem( NJD_RESOLUTION_VGA,
NJD_FRAMEBUFFER_MODE_RGB565, 1 );

    :

    /* Reserve texture memory area with dummy */
    buff = njMalloc(512*512*2);
```



```
512x512 */
/* Set color the same as the frame buffer color, and size to
njSetTextureInfo(&info,buff,NJD_TEXFMT_RECTANGLE|NJD_TEXFMT_RGB_565,512,512);
njSetTextureName(&texname[0],&info,0,NJD_TEXATTR_TYPE_MEMORY|
NJD_TEXATTR_GLOBALINDEX);

njInitTexture( tex, 100 );
njLoadTexture(&texlist);

/* Reserved dummy area can be released after the njLoadTexture
function */
njFree(buff);
}

 Sint32 njUserMain(void)
{
:

/* Draw models etc. */
njDrawObject( OBJECT );

:

njSetTexture(&texlist);
/* Renders to the texture number 0. */
njRenderTextureNum(0);

/* Draw using rendered textures. */
njDrawTexture( poly, 4, 0,TRUE);

:
}
```

## NOTES

- Rendering to the texture area results in an additional rendering to the frame buffer.
- Refer to texture document for more details.

## RELATED TOPICS

[njRenderTextureNumG](#)

# njRenderTextureNumG

Renders at the Texture Area.

---

## FORMAT

```
#include <Ninja.h>
void njRenderTextureNumG(globalIndex);
Uint32 globalIndex
```

## PARAMETERS

**globalIndex**

global index number

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Renders at the texture specified by globalIndex.

## EXAMPLE

## NOTES

## RELATED TOPICS

[njRenderTextureNum](#)

## **njSetRenderWidth**

Sets Stride Value.

---

### **FORMAT**

```
#include <Ninja.h>
void njSetRenderWidth(nWidth);
Uint32 nWidth
```

### **PARAMETERS**

**nWidth**

stride value

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Sets stride value when the stride texture format is used.
- When stride texture is specified for rendering texture, it sets texture width size if it is smaller than rendering area . And, it sets the rendering area width size if it is smaller than the texture size. Values set are multiples of 32 (up to 992).

### **EXAMPLE**

### **NOTES**

### **RELATED TOPICS**

`njRenderTextureNum`

`njRenderTextureNumG`





## *12. Sprite Functions*

---

### **Contents**

njDrawSprite2D .....	Draws 2D sprite. ....	NLS-396
njDrawSprite3D .....	Draws 3D sprite. ....	NLS-400

## njDrawSprite2D

Draws 2D sprite.

### FORMAT

```
#include <Ninja.h>
void  njDrawSprite2D( *sp, n, pri, attr )
NJS_SPRITE  *sp
Int          n
Float        pri
Uint32       attr
```

### PARAMETERS

**\*sp**

Pointer for Sprite structure

**n**

Sprite number

**pri**

Priority

**attr**

Attribute

### RETURN VALUE

None

### ERROR VALUE

None

## FUNCTION

- Draws 2D sprite.
- Sprite can draw textures and its animations simply.
- Priority sets the Z value.
- Back is minus direction and its range is -1.0 thru -65535.0.
- The followings can be set as attribute:

<b>NJD_SPRITE_ANGLE</b>	Validate texture rotation.
<b>NJD_SPRITE_COLOR</b>	The value set to <code>njSetConstantMaterial()</code> is used as color data.
<b>NJD_SPRITE_HFLIP</b>	Reverse texture horizontally (L<->R).
<b>NJD_SPRITE_VFLIP</b>	Reverse texture vertically (Up<->Down).
<b>NJD_SPRITE_HVFLIP</b>	Reverse texture horizontally and vertically.
<b>NJD_SPRITE_ALPHA</b>	Set if translucent is used.

### Sprite sructure

```
typedef struct {
    NJS_POINT3  p;          /* point          */
    NJS_VECTOR  v;          /* normal         */
    Float       sx,sy;      /* scale         */
    Angle       ang;        /* angle         */
    NJS_TEXLIST *tlist;     /* texture list  */
    NJS_TEXANIM *tanim;     /* animation list */
} NJS_SPRITE;
```

### **p**

sets drawing point.

p.x and p.y : set screen coordinate (upper left origin).

### **v**

\*is not valid in the current version.

### **sx,sy**

specify the magnifying rate in H and V.

### **ang**

sets the rotate angle.

It is valid only when NJD\_SPRITE\_ANGLE is set.

### **tlist**

sets texture list of all textures used for Sprite.

The texture list must be loaded by njLoadTexture() beforehand.

### **tanim**

sets texture animation patterns.

The order of this structure accommodates texture numbers.

### **Texture animation structure**

```
typedef struct {  
    Sint16    sx,sy;           /* size      */  
    Sint16    cx,cy;           /* center    */  
    Sint16    u1,v1;           /* tex1      */  
    Sint16    u2,v2;           /* tex2      */  
    Sint16    texid;           /* texture id */  
    Sint16    attr;            /* attribute  */  
} NJS_TEXANIM;
```

### **sy,sy**

specify sprite size (irrelevant to texture size)

### **cx,cy**

set the center coordinate for the texture.

specify by relative point from upper left.

This point corresponds to drawing point and the center of rotation.

### **u1,v1**

set texture upper-left UV coordinate.

0 - 255 must be specified irrelevant with texture size.

### **u2,v2**

set texture bottom-right UV coordinate.

0 - 255 must be specified irrelevant with texture size.

### **texid**

specify which texture within the list is used.

### **attr**

\*invalid at the current version.



## EXAMPLE

- Use two 256x256 textures.
- Divide each texture into four patterns and make 8 texture animation patterns.

```
NJS_TEXANIM  anim[] = {
    { 128,128, 64, 64,  0,  0,127,127,  0, 0 },
    { 128,128, 64, 64,128,  0,255,127,  0, 0 },
    { 128,128, 64, 64,  0,128,127,255,  0, 0 },
    { 128,128, 64, 64,128,128,255,255,  0, 0 },
    { 128,128, 64, 64,  0,  0,127,127,  1, 0 },
    { 128,128, 64, 64,128,  0,255,127,  1, 0 },
    { 128,128, 64, 64,  0,128,127,255,  1, 0 },
    { 128,128, 64, 64,128,128,255,255,  1, 0 },
};

main() {
    NJS_SPRITE  sprite;

    Initial settings
    :
    :

    sprite.tlist = &texlist;
    sprite.tanim = anim;
    sprite.ang = 0x4000;
    sprite.sx = 1.0f;
    sprite.sy = 1.0f;
    sprite.p.x = 320.0f;
    sprite.p.y = 240.0f;

    njLoadTexture( &texlist );
    njDrawSprite2D( &sprite, 2, -150.f, NJD_SPRITE_ANGLE );

    .....
}
```

## NOTES

## RELATED TOPICS

`njDrawSprite3D()`  
`njSetConstantMaterial()`  
`njLoadTexture()`

## njDrawSprite3D

Draws 3D sprite.

### FORMAT

```
#include <Ninja.h>
void njDrawSprite3D( *sp, n, attr )
NJS_SPRITE *sp
Int n
Uint32 attr
```

### PARAMETERS

**\*sp**

Pointer for sprite structure

**n**

Sprite number

**attr**

Attribute

### RETURN VALUE

None

### ERROR VALUE

None

### FUNCTION

- Draws 3D sprite.
- It can draw texture and its animation.
- Draws 3D sprite across the XY plane.  
It is almost the same as 2D drawing except it draws in 3D.
- Attribute valid only for 3D:

<b>NJD_SPRITE_SCALE</b>	Magnification/Reduction only. Always front to the screen.
-------------------------	--

**EXAMPLE**

- Use two 256x256 textures and divide each into 4 patterns, making 8 texture animation patterns.

```

NJS_TEXANIM  anim[] = {
    { 128,128, 64, 64,  0,  0,127,127,  0, 0 },
    { 128,128, 64, 64,128,  0,255,127,  0, 0 },
    { 128,128, 64, 64,  0,128,127,255,  0, 0 },
    { 128,128, 64, 64,128,128,255,255,  0, 0 },
    { 128,128, 64, 64,  0,  0,127,127,  1, 0 },
    { 128,128, 64, 64,128,  0,255,127,  1, 0 },
    { 128,128, 64, 64,  0,128,127,255,  1, 0 },
    { 128,128, 64, 64,128,128,255,255,  1, 0 },
};

main() {
    NJS_SPRITE  sprite;

    Initial settings

    sprite.tlist = &texlist;
    sprite.tanim = anim;
    sprite.ang = 0;
    sprite.sx = 1.0f;
    sprite.sy = 1.0f;
    sprite.p.x = 0.0f;
    sprite.p.y = 0.0f;
    sprite.p.z = 0.0f;

    njInitSystem( NJD_RESOLUTION_640x480, 1 );
    njInitMatrix( matrix, 128 );
    njInit3D( vbuf, 1024, abuf, fbuf, 32 );
    njInitView( &view );
    njCreateLight( &light, NJD_DIR_LIGHT );
    njSetView( &view );
    njInitTexture( tex, 100 );

    njLoadTexture( &texlist );

    while(1) {
        njClearMatrix();

        njTranslate( NULL, 0.f, 0.f, -400.f );
        njDrawSprite3D( &sprite, 2,

        sprite.angle += 100;

        .....
    }
}

```





## *13. Debugging Functions*

---

### **Contents**

njPrintB .....	Displays a value in binary notation. ....	NLS-404
njPrintC .....	Displays a character string. ....	NLS-406
njPrintColor .....	Specifies the character color. ....	NLS-407
njPrintD .....	Displays a value in decimal notation. ....	NLS-408
njPrintF .....	Displays a floating point decimal value. ....	NLS-410
njPrintH .....	Displays a value in decimal notation. ....	NLS-412
njFrameBufferBmp .....	Converts Frame Buffer To Bit Map Image. ....	NLS-414
njPrintSize .....	Specifies Character Size. ....	NLS-415

## njPrintB

Displays a value in binary notation.

---

### FORMAT

```
#include <Ninja.h>
void njPrintB(loc,val,digit);
Int loc
Uint32 val
Int digit
```

### PARAMETERS

**loc**

display position

**val**

value

**digit**

number of output digits

### RETURN VALUE

None

### FUNCTION

- Displays the specified value on screen as a binary number.

### EXAMPLE

The following outputs the value of 100 as a binary number at location 10,10.

```
njPrintB(NJM_LOCATION(10,10),100,10);
```

## NOTES

- The display position is specified by NJM\_LOCATION(x,y).
- x,y specify a character position in a row.
- If the value is greater than the number of output digits, the value following the number of output digits is output.

## RELATED TOPICS

`njPrintColor`

`njPrintC`

`njPrintD`

`njPrintH`

`njPrintF`

# njPrintC

Displays a character string.

---

## FORMAT

```
#include <Ninja.h>

void njPrintC(loc,*s);

Int loc

Uint8 *s
```

## PARAMETERS

**loc**  
display position

**\*s**  
character string to be output

## RETURN VALUE

None

## FUNCTION

- Displays a character string at the specified screen location.

## EXAMPLE

```
njPrintC(NJM_LOCATION(10,10) , "NINJA TEST" );
```

## NOTES

- The display position is specified by NJM\_LOCATION(x,y).
- x,y specify a character position in a row.

## RELATED TOPICS

[njPrintColor](#)

[njPrintD](#)

[njPrintB](#)

[njPrintH](#)

[njPrintF](#)



## njPrintColor

Specifies the character color.

---

### FORMAT

```
#include <Ninja.h>
void njPrintColor(c);
Uint32 c
```

### PARAMETERS

**c**

color value

### RETURN VALUE

None

### FUNCTION

- Specifies the color of debug printing. The color is entered as an RGB value.
- The first 8 bits of the A value are invalid.

### EXAMPLE

Set the character to red, 255.

```
njPrintColor(0x00FF0000);
```

### NOTES

- When njPrintColor is not executed, the default debug print color is white (0x00FFFFFF).
- After executing njPrintColor, the specified color is used for all debug printing.

### RELATED TOPICS

[njPrintC](#)

[njPrintD](#)

[njPrintB](#)

[njPrintH](#)

[njPrintF](#)

# **njPrintD**

Displays a value in decimal notation.

---

## **FORMAT**

```
#include <Ninja.h>
```

```
void njPrintD(loc,val,digit);
```

Int loc

Int val

Int digit

## **PARAMETERS**

**loc**

display position

**val**

value

**digit**

number of output digits

## **RETURN VALUE**

None

## **FUNCTION**

- Displays the specified value on screen as a decimal number

## **EXAMPLE**

The following outputs the value of 100 as 3 digits at location 10,10.

```
njPrintD(NJM_LOCATION(10,10),100,3);
```

## NOTES

- The display position is specified by NJM\_LOCATION(x,y).
- x,y specify a character position in a row.
- If the value is greater than the number of output digits, the value following the number of output digits is output.

## RELATED TOPICS

`njPrintColor`

`njPrintC`

`njPrintD`

`njPrintH`

`njPrintF`

## **njPrintf**

Displays a floating point decimal value.

---

### **FORMAT**

```
#include <Ninja.h>
void njPrintD(loc,val,digit);
Int loc
Float val
Int digit
```

### **PARAMETERS**

**loc**

display position

**val**

value

**digit**

number of output digits

### **RETURN VALUE**

None

### **FUNCTION**

- Displays the specified value on screen as a floating point decimal number.

### **EXAMPLE**

The following outputs the value 3.14 as 10 digits at location 10,10.

```
njPrintf(NJM_LOCATION(10,10),3.14,10);
```

## NOTES

- The display position is specified by NJM\_LOCATION(x,y).
- x,y specify a character position in a row.
- If the value is greater than the number of output digits, the value following the number of output digits is output.

## RELATED TOPICS

`njPrintColor`

`njPrintC`

`njPrintD`

`njPrintB`

`njPrintH`

# njPrintH

Displays a value in decimal notation.

---

## FORMAT

```
#include <Ninja.h>
void njPrintH(loc,val,digit);
Int loc
Uint32 val
Int digit
```

## PARAMETERS

**loc**

display position

**val**

value

**digit**

number of output digits

## RETURN VALUE

None

## FUNCTION

- Displays the specified value on screen as a hexadecimal number.

## EXAMPLE

The following outputs the value of 100 as a 3 hexadecimal digits at location 10,10.

```
njPrintH(NJM_LOCATION(10,10),100,3);
```

## NOTES

- The display position is specified by NJM\_LOCATION(x,y).
- x,y specify a character position in a row.
- If the value is greater than the number of output digits, the value following the number of output digits is output.

## RELATED TOPICS

`njPrintColor`

`njPrintC`

`njPrintD`

`njPrintB`

`njPrintF`

## **njFramebufferBmp**

Converts Frame Buffer To Bit Map Image.

---

### **FORMAT**

```
#include <Ninja.h>
void njFramebufferBmp(filename);
Uint8 *filename
```

### **PARAMETERS**

**filename**

file name

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- Changes the context of frame buffer into 24bit texture image.
- Currently, it applies only for flame#0. So using this function while frame#0 is drawn creates an image in the middle of rendering. (will be modified in future)

### **EXAMPLE**

```
/* Converts the frame buffer context to a bitmap image as frame.bmp. */
njFramebufferBmp("frame.bmp");
```

### **NOTES**

- Specification will be modified so that the current frame will be converted to a bitmap texture. It can be used for debugging purpose.

### **RELATED TOPICS**



# **njPrintSize**

Specifies Character Size.

---

## **FORMAT**

```
#include <Ninja.h>
void njPrintSize(size);
Uint16 size
```

## **PARAMETERS**

**size**

character size

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

Specifies the character size.

## **EXAMPLE**

Sets the character size 16x16 and output.

```
njPrintSize(16);
```

## **NOTES**

- If specified sizes are varied in a screen, the character positions will not be aligned.

## **RELATED TOPICS**

njPrintB  
njPrintColor  
njPrintC  
njPrintD  
njPrintH  
njPrintF





# 14. *Special Effects Functions*

---

## Contents

njExcuteFade .....	Excutes the fade effect. ....	NLS-418
njFadeDisable .....	Disables the fade effect. ....	NLS-420
njFadeEnable.....	Enables the fade effect. ....	NLS-422
njFogDisable .....	Disables the fog effect. ....	NLS-424
njFogEnable.....	Enables the fog effect. ....	NLS-426
njGenerateFogTable .....	Creates Fog Table. ....	NLS-428
njGenerateFogTable2 .....	Creates Fog Table and Sets Density. ....	NLS-430
njGenerateFogTable3 .....	Creates Fog Table and Sets Density. ....	NLS-432
njSetFadeColor .....	Specifies the fade color. ....	NLS-434
njSetFogColor .....	Specifies the fog color. ....	NLS-436
njSetFogDensity .....	Specifies the fog density. ....	NLS-438
njSetFogTable .....	Sets a user-defined fog table. ....	NLS-440

## njExcuteFade

Excutes the fade effect.

### FORMAT

```
#include <Ninja.h>
void njExcuteFade(f)
Float f
```

### PARAMETERS

**unsigned char f**

fade ratio (0x00~0xff)

### RETURN VALUE

None

### FUNCTION

- Executes a fade.
- The color remains unchanged if fade ratio is 0x00.  
With 0xff, fading goes all the way to the specified color.

### EXAMPLE

The following fades out to green.

```
Float fade_val
```

```
njSetFadeColor(0xff00ff00);
njExcuteFade(fade_val);
njFadeEnable();
```

```

                                :
Drawing of models, etc.      :
                                :
```

```
njFadeDisable();
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

---

## NOTES

- It can't be used with Fog feature against the same object.

## RELATED TOPICS

`njSetFadeColor()`

`njFadeExecute()`

`njFadeEnable()`

`njFadeDisable()`

# **njFadeDisable**

Disables the fade effect.

## **FORMAT**

```
#include <Ninja.h>
void njFadeDisable()
```

## **PARAMETERS**

None

## **RETURN VALUE**

None

## **FUNCTION**

- Disables the fade effect.

## **EXAMPLE**

Fading out to Green.

```
Float fade_val
```

```
njSetFadeColor(0xff00ff00);
njExcuteFade(fade_val);
njFadeEnable();
```

```
                                :
Drawing models etc. (fade valid.)
                                :

njFadeDisable();

                                :
Drawing models etc. (fade valid.)
                                :
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

---

## NOTES

- It can't be used with Fog feature against the same object.

## RELATED TOPICS

`njSetFadeColor()`

`njFadeExecute()`

`njFadeEnable()`

`njFadeDisable()`

## njFadeEnable

Enables the fade effect.

### FORMAT

```
#include <Ninja.h>
void njfadeEnable()
```

### PARAMETERS

None

### RETURN VALUE

None

### FUNCTION

- Enables the fade effect.

### EXAMPLE

Fading out to Green.

```
Float fade_val
```

```
njSetFadeColor(0xff00ff00);
njExcuteFade(fade_val);
njFadeEnable();
```

```
                                :
                                drawing models etc.
                                :
```

```
njFadeDisable();
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

---



## NOTES

- It can't be used with Fog feature against the same object.

## RELATED TOPICS

`njSetFadeColor()`

`njFadeExecute()`

`njFadeEnable()`

`njFadeDisable()`

# **njFogDisable**

Disables the fog effect.

## **FORMAT**

```
#include <Ninja.h>
void njFogDisable()
```

## **PARAMETERS**

None

## **RETURN VALUE**

None

## **FUNCTION**

- Disables the fog effect.

## **EXAMPLE**

```
NJS_FOG_TABLE fog;

njSetFogColor(0xffffffff);
njSetFogDensity(0xff07);
njGenerateFogTable(fog);
njSetFogTable();
njFogEnable();

:
Drawing models etc. (Fade valid)
:

njFogDisable();

:
Drawing models etc. (Fade valid)
:
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

---

## NOTES

- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFogColor()`

`njSetFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`

## njFogEnable

Enables the fog effect.

## FORMAT

```
#include <Ninja.h>
void njFogEnable()
```

## PARAMETERS

None

## RETURN VALUE

None

## FUNCTION

- Creates white fog for  $w = -1.f$  thru  $-255.f$ .

## EXAMPLE

```
NJS_FOG_TABLE fog;

njSetFogColor(0xffffffff);
njSetFogDensity(0xff07);
njGenerateFogTable(fog);
njSetFogTable();
njFogEnable();
```

Drawing models etc.

```
njFogDisable();
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

## NOTES

- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFogColor()`

`njSetFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`

## njGenerateFogTable

Creates Fog Table.

## FORMAT

```
#include <Ninja.h>

void njGenerateFogTable(fog)
NJS_FOG_TABLE    fog
```

## PARAMETERS

## NJS\_FOG\_TABLE fog

Fog table to be made

## RETURN VALUE

None

## FUNCTION

- Sets each value of the fog table from 0.f to 1.f in linear.

## EXAMPLE

Creates white fog from front -1 to back -255.

```
NJS_FOG_TABLE fog;

njSetFogColor(0xffffffff);
njSetFogDensity(0xff07);
njGenerateFogTable(fog);
njSetFogTable();
njFogEnable();
```

Drawing models etc.

```

njFogDisable();

```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

## NOTES

- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFogColor()`

`njSetFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`

## njGenerateFogTable2

Creates Fog Table and Sets Density.

## FORMAT

```
#include <Ninja.h>

void njGenerateFogTable2(fog, f)
NJS_FOG_TABLE    fog
Float            f
```

## PARAMETERS

## NJS\_FOG\_TABLE fog

Fog tabel to be created

**Float f**

Fog back boundary

## RETURN VALUE

None

## FUNCTION

- Makes fog occur from front  $255/f$  to back  $f$ .

## EXAMPLE

Makes white fog from front -255/4000 to back -4000.

```
NJS_FOG_TABLE fog;

njSetFogColor(0xffffffff);
njGenerateFogTable2(fog, -4000,f);
njSetFogTable();
njFogEnable();
```

Drawing models etc.

```

njFogDisable();

```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.



## NOTES

- Fog density is set automatically.
- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFogColor()`

`njSetFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`

## njGenerateFogTable3

Creates Fog Table and Sets Density.

---

### FORMAT

```
#include <Ninja.h>

void njGenerateFogTable3(fog, n, f)
NJS_FOG_TABLE    fog
Float            n
Float            f
```

### PARAMETERS

**NJS\_FOG\_TABLE fog**

Fog table to be created

**Float n**

Fog front boundary

**Float f**

Fog back boundary

### RETURN VALUE

None

### FUNCTION

- Creates fog from front n to back f.



## njSetFadeColor

Specifies the fade color.

## FORMAT

```
#include <Ninja.h>

void njSetFadeColor(c)
Uint32 c
```

## PARAMETERS

Uint32 c

## RETURN VALUE

None

## FUNCTION

- Specifies the fade color.

## EXAMPLE

The following fades out to green.

```
Float fade_val;

njSetFadeColor(0xff00ff00);
njExcuteFade(fade_val);
njFadeEnable();
```

Drawing models etc.

```
njFadeDisable();
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

## NOTES

- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFadeColor()`

`njExcuteFade()`

`njFadeEnable()`

`njFadeDisable()`

## njSetFogColor

Specifies the fog color.

## FORMAT

```
#include <Ninja.h>

void njSetFogColor(c)
Uint32 c
```

## PARAMETERS

**Uint32 c**

color used for fog effect (ARGB)

## RETURN VALUE

None

## FUNCTION

- Specifies the fog color.

## EXAMPLE

The following generates green fog.

```
NJS_FOG_TABLE fog;

njSetFogColor(0xffffffff);
njFogDensity(0xff07);
njGenerateFogTable(fog);
njSetFogTable(fog);
njFogEnable();
```

Drawing of models, etc.

```

njFogDisable();

```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

## NOTES

- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFogColor()`

`njFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`

## njSetFogDensity

Specifies the fog density.

## FORMAT

```
#include <Ninja.h>

void njSetFogDensity(density)
    Uint32 density
```

## PARAMETERS

## UInt32 density

value of sinedensity

## RETURN VALUE

None

## FUNCTION

- Specifies the fog density.

## EXAMPLE

- Creates white fog from  $w = -1.f$  to  $-255.f$ .

```
NJS_FOG_TABLE fog;

njSetFogColor(0xffffffff);
njFogDensity(0xff07);
njGenerateFogTable(fog);
njSetFogTable(fog);
njFogEnable();
```

Drawing models etc.

```
njFogDisable();
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.



## NOTES

- It cannot be used with the fade feature against the same object.
- The example above explains only the flow of the fog feature. It cannot be executed as is.
- The fog density value is specified as 16 bit: upper 8bit = mantissa; lower 8bit= exponent.
- Range for the fog is  $1/w = 1/\text{density} \dots 256/\text{density}$ . For example, when the fog reaches the value of w, 65536: density = 65536 = 0x8010, and the range for the fog is w=65536 to w=256.
- Fog color that is out of range is clumped.

## RELATED TOPICS

`njSetFogColor()`

`njFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`

# njSetFogTable

Sets a user-defined fog table.

## FORMAT

```
#include <ninja.h>

void njSetFogTable(fog)
NJS_FOG_TABLE fog
```

## PARAMETERS

**NJS\_FOG\_TABLE fog**

fog table

## RETURN VALUE

None

## FUNCTION

- Sets a user-defined fog table.

## EXAMPLE

Makes green fog from front -10000 to back -40000 in linear.

```
NJS_FOG_TABLE fog;
njGenerateFogTable(fog, -10000, -40000);
njSetFogTable(fog);
njSetFogColor(0xff00ff00);
njFogEnable();
```

Drawing models etc.

```
njFogDisable();
```



**NOTE:** The example above only explains the flow of the fog feature, and therefore, it is not for actual use.

---

## NOTES

- It can't be used with Fade feature against the same object.

## RELATED TOPICS

`njSetFogColor()`

`njFogDensity()`

`njSetFogTable()`

`njFogEnable()`

`njFogDisable()`

`njGenerateFogTable()`

`njGenerateFogTable2()`

`njGenerateFogTable3()`





## 15. Motion Functions

---

### Contents

njAction.....	Drawing motion. ....	NLS-422
njActionLink .....	Links motions. ....	NLS-424
njDrawMotion .....	Drawing Motion. ....	NLS-426
njDrawMotionLink .....	Links motions. ....	NLS-428
njDrawShapeMotion.....	Executes motion that includes shapes. ....	NLS-430
njDrawShapeMotionLink.....	Links motions that include shapes. ....	NLS-432
njFastAction .....	Draws Motion. ....	NLS-434
njFastActionLink .....	Links motions. ....	NLS-436
njFastDrawMotion .....	Draws Motion. ....	NLS-438
njFastDrawMotionLink .....	Links motions. ....	NLS-440
njFastDrawShapeMotion.....	Executes motion that includes shapes. ....	NLS-442
njFastDrawShapeMotionLink .....	Links motions that include shapes. ....	NLS-444
njSetMotionCallback.....	Registering motion callback routine. ....	NLS-446

# **njAction**

Drawing motion.

## **FORMAT**

```
#include <Ninja.h>
void  njAction( *action, frame )
NJS_ACTION *action
Float  frame
```

## **PARAMETERS**

### **action**

Pointer to the action structure

### **frame**

Frame number

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Draws object structure with motion - action structure.
- Floating point can be used for frame number, so it can supplement motion data.
- The range for frame number is 0 thru less than 10 (9.9999) when the frame number of a motion is 10.
- For example, suppose the frame number is 0.5. It will supplement the values between 0th frame and 1st frame and draw.
- Supplements for 9 thru 10(9.9999) would be performed between the last frame and the first frame.

## EXAMPLE

```
#define ACTION    action_man
extern NJS_ACTION ACTION[];

void main() {
    Float ff=0.0f;

    njInitSystem( NJD_RESOLUTION_640x480, 1 );
    njInitMatrix( matrix, 128 );
    njInit3D( vbuf, 1024, abuf, fbuf, 32 );
    njInitView( &view );
    njCreateLight( &light, NJD_DIR_LIGHT );
    njSetView( &view );

    while(1) {
        njClearMatrix();
        njAction( ACTION,ff );

        ff+=0.2f;
        if( ff >= (ACTION->motion->nbFrame) )
            ff-=ACTION->motion->nbFrame;

        njWaitVSync();
    }
}
```

## NOTES

- As for drawing, it is completely the same as model drawing, except that it has motion.
- Therefore, settings for njControl3D are the same.

## RELATED TOPICS

[njInit3D\(\)](#)

[njDrawModel\(\)](#)

[njDrawObject\(\)](#)

[njControl3D\(\)](#)

[njDrawMotion\(\)](#)

## **njActionLink**

**Links motions.**

---

### **FORMAT**

```
#include <Ninja.h>

void njActionLink( actionlink, frame )

NJS_ACTION_LINK *actionlink

Float frame
```

### **PARAMETERS**

**actionlink**

Pointer to action link structure

**frame**

Frame number (from 0 to 1)

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- This function links two motion structures.
- The frames in the motion link structure are the link source and link destination motion frames.
- When the frame parameter is 0, the frame is part of the motion of the link source frame; when the frame parameter is 1, the frame is part of the motion of the link destination frame.
- Other values ( $0 < ff < 1$ ) interpolate the motion between the link destination frame and the link source frame.



## EXAMPLE

```
#include <NINJA.H>

#define OBJECT          object_sample
#define MOTION1         motion_sample1 /* Described in nam */
#define MOTION2         motion_sample2 /* Described in nam */
#define STEPS           60

extern NJS_MOTION       MOTION1[];
extern NJS_MOTION       MOTION2[];

NJS_MOTION_LINK motionlink;
NJS_ACTION_LINK actionlink;

.....
actionlink.motionlink = &motionlink;
actionlink.object = OBJECT;
motionlink.motion[0] = MOTION1; /* link source */
motionlink.motion[1] = MOTION2; /* link destination */
/* interpolate 1.5 frames of MOTION2 from the final frame of MOTION1 */
motionlink.frame[0] = MOTION1->nbFrame-1;
motionlink.frame[1] = 1.5;

.....
njActionLink( &actionlink, ff/STEPS ); /* Specify from 0 - 1 for frames */

ff+=0.2f;
if( ff >= STEPS - 1 ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion has been added. In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njDrawObject, so the light source is normal light.

## RELATED TOPICS

Related Items

[njDrawMotionLink\(\)](#)

[njFastActionLink\(\)](#)

[njFastDrawMotionLink\(\)](#)

# njDrawMotion

Drawing Motion.

---

## FORMAT

```
#include <Ninja.h>
void njDrawMotion( *object, *motion, frame )
NJS_OBJECT *object
NJS_MOTION *motion
Float      frame
```

## PARAMETERS

### **object**

Pointer for the object structure

### **motion**

Pointer for the motion structure

### **frame**

Frame number

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- Draws motion following motion data.
- Floating point can be used for frame number, so it can supplement motion data.
- The range for frame number is 0 thru less than 10 (9.9999) when the frame number of a motion is 10.
- For example, suppose the frame number is 0.5. It will supplement the values between 0th frame and 1st frame and draw.
- Supplements for 9 thru 10(9.9999) would be performed between the last frame and the first frame.

## EXAMPLE

```
#define OBJECT    object_man
#define MOTION    motion_man
extern NJS_OBJECT OBJECT[];
extern NJS_MOTION MOTION[];

void main() {
    Float ff=0.0f;

    njInitSystem( NJD_RESOLUTION_640x480, 1 );
    njInitMatrix( matrix, 128 );
    njInit3D( vbuf, 1024, abuf, fbuf, 32 );
    njInitView( &view );
    njCreateLight( &light, NJD_DIR_LIGHT );
    njSetView( &view );

    while(1) {
        njClearMatrix();
        njDrawMotion( OBJECT, MOTION, ff );

        ff+=0.2f;
        if( ff >= (MOTION->nbFrame) )

            njWaitVSync();

        ff-=MOTION->nbFrame;
    }
}
```

## NOTES

- This is the same as njAction() except for that it is separated into object and motion.

## RELATED TOPICS

[njInit3D\(\)](#)

[njDrawModel\(\)](#)

[njDrawObject\(\)](#)

[njControl3D\(\)](#)

[njAction\(\)](#)

# njDrawMotionLink

Links motions.

---

## FORMAT

```
#include <Ninja.h>
void njDrawMotionLink( object, motionlink, frame )
NJS_OBJECT      *object
NJS_MOTION_LINK *motionlink
Float           frame
```

## PARAMETERS

None

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- This function links two motion structures.
- The frames in the motion link structure are the link source and link destination motion frames.
- When the frame parameter is 0, the frame is part of the motion of the link source frame; when the frame parameter is 1, the frame is part of the motion of the link destination frame.
- Other values ( $0 < ff < 1$ ) interpolate the motion between the link destination frame and the link source frame.

## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample
#define MOTION1    motion_sample1 /* described in nam */
#define MOTION2    motion_sample2 /* described in nam */
#define STEPS      60

extern NJS_MOTION  MOTION1[];
extern NJS_MOTION  MOTION2[];

NJS_MOTION_LINK  motionlink;

.....
motionlink.motion[0] = MOTION1; /* link source */
motionlink.motion[1] = MOTION2; /* link destination */
/* interpolate 1.5 frames of MOTION2 from the final frame of MOTION1 */
motionlink.frame[0] = MOTION1->nbFrame-1;
motionlink.frame[1] = 1.5;

.....
njDrawMotionLink( OBJECT, &motionlink, ff/STEPS ); /* Specify from 0 - 1 for frames */

ff+=0.2f;
if( ff >= STEPS - 1 ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion has been added. In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njDrawObject, so the light source is normal light.

## RELATED TOPICS

njActionLink()  
njFastActionLink()  
njFastDrawMotionLink()

## **njDrawShapeMotion**

Executes motion that includes shapes.

---

### **FORMAT**

```
#include <Ninja.h>
void njInitShape( Float *buf )
Float      *buf
```

### **PARAMETERS**

**\*buf**

Shape buffer

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- This function sets a temporary buffer for shapes.
- Assuming **N** is the maximum number of vertices in the models that are used as shapes within all objects, the number of buffers needed is  $N \times 3$  for shapes that consist only of vertices or normal lines, and  $N \times 6$  for shapes that consist of both.
- For example, given the objects **Person 1** and **Person 2**, in which only vertex shapes are used, then if the model for the **Face of Person 1** uses 1000 vertices, and this is the largest number of vertices within the object, then this function is specified as **buf [ 3000 ]**.

## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample /* described within NJA (NAM) file */
#define MOTION    motion_sample /* described within NAM file */
#define SHAPE     shape_sample  /* described within NAS file */
extern NJS_OBJECT OBJECT[];
extern NJS_MOTION MOTION[];
extern NJS_MOTION SHAPE[];
float  buf[6000];
float  ff=0.f;

.....
InitShape(buf);

.....
njDrawShapeMotion( OBJECT,MOTION,SHAPE,ff );

ff+=0.2f;
if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
.....
```

## NOTES

This function must be set for the following functions that draw shapes (vertex animation) (in which the shape motion parameter is not NULL): `njDrawShapeMotion`, `njFastDrawShapeMotion`, `njDrawShapeMotionLink`, and `njFastDrawShapeMotionLink`.

## RELATED TOPICS

`njDrawShapeMotion()`  
`njFastDrawShapeMotion()`  
`njDrawShapeMotionLink()`  
`njFastDrawShapeMotionLink()`

## **njDrawShapeMotionLink**

Links motions that include shapes.

---

### **FORMAT**

```
#include <Ninja.h>
void njDrawShapeMotionLink( object, motionlink, shapelink, frame )
NJS_OBJECT      *object
NJS_MOTION_LINK *motionlink
NJS_MOTION_LINK *shapelink
Float           frame
```

### **PARAMETERS**

#### **object**

Pointer to object structure

#### **motionlink**

Pointer to motion link structure

#### **shapelink**

Pointer to motion link structure (vertex animation)

#### **frame**

Frame number (from 0 to 1)

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- This function links two motion structures.
- The frames in the motion link structure are the link source and link destination motion frames.
- When the frame parameter is 0, the frame is part of the motion of the link source frame; when the frame parameter is 1, the frame is part of the motion of the link destination frame.
- Other values ( $0 < ff < 1$ ) interpolate the motion between the link destination frame and the link source frame.
- This also applies to shapes (vertex animation).



## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample
#define MOTION1    motion_sample1 /* described within NAM file */
#define MOTION2    motion_sample2 /* described within NAM file */
#define SHAPE1    shape_sample1  /* described within NAS file */
#define SHAPE2    shape_sample2  /* described within NAS file */
#define STEPS      60

extern NJS_MOTION MOTION1[];
extern NJS_MOTION MOTION2[];
extern NJS_MOTION SHAPE1[];
extern NJS_MOTION SHAPE2[];

NJS_MOTION_LINK motionlink;
NJS_MOTION_LINK shapelink;

.....
motionlink.motion[0] = MOTION1; /* link source */
motionlink.motion[1] = MOTION2; /* link destination */
shapelink.motion[0] = SHAPE1; /* link source */
shapelink.motion[1] = SHAPE2; /* link destination */
/* interpolate 1.5 frames of MOTION2 from the final frame of MOTION1 */
motionlink.frame[0] =
shapelink.frame[0] = MOTION1->nbFrame-1;
motionlink.frame[1] =
shapelink.frame[1] = 1.5;

.....
njDrawShapeMotionLink( OBJECT, &motionlink, &shapelink, ff/STEPS );
/* Specify from 0 - 1 for frames */

ff+=0.2f;
if( ff >= STEPS - 1 ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion and shapes (vertex animation) have been added.

In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njDrawObject, so the light source is normal light.

## RELATED TOPICS

`njInitShape()`

`njFastDrawShapeMotionLink()`

# **njFastAction**

Draws Motion.

## **FORMAT**

```
#include <Ninja.h>
void  njFastAction( action, frame )
NJS_ACTION  *action
Float      frame
```

## **PARAMETERS**

**action**

pointer for the action structure

**frame**

frame number

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- This function draws an object structure with motion - an action structure.
- Because floating-point data can be used for frame numbers, the motion data can be interpolated.
- In the case of a motion consisting of 10 frames, the frame numbers can range from 0 to just under 10 (i.e., 9.9999).
- For example, if the frame number is 0.5, the values used for drawing are interpolated as exactly halfway between the values for frame 0 and the values for frame 1.
- Interpolation from 9 to just under 10 (9.9999) becomes interpolation between the last frame and the first frame.

## EXAMPLE

```
#include <NINJA.H>

#define ACTION    action_sample /* described in nam */
extern NJS_ACTION ACTION[];

.....
njFastAction( ACTION,ff );

ff+=0.2f;
if( ff >= (ACTION->motion->nbFrame-1) ) ff=0.f;
```

## NOTES

- The drawing performed by this function is identical to model drawing, except that motion has been added. In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njFastDrawObject, so the light source is bright light.

## RELATED TOPICS

[njFastDrawMotion\(\)](#)

[njAction\(\)](#)

[njDrawMotion\(\)](#)

# **njFastActionLink**

**Links motions.**

---

## **FORMAT**

```
#include <Ninja.h>
void njFastActionLink( actionlink, frame )
NJS_ACTION_LINK *actionlink
Float           frame
```

## **PARAMETERS**

### **actionlink**

Pointer to action link structure

### **frame**

Frame number (from 0 to 1)

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- This function links two motion structures.
- The frames in the motion link structure are the link source and link destination motion frames.
- When the frame parameter is 0, the frame is part of the motion of the link source frame; when the frame parameter is 1, the frame is part of the motion of the link destination frame.
- Other values ( $0 < ff < 1$ ) interpolate the motion between the link destination frame and the link source frame.

## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample
#define MOTION1    motion_sample1 /* described in nam */
#define MOTION2    motion_sample2 /* described in nam */
#define STEPS      60

extern NJS_MOTION  MOTION1[];
extern NJS_MOTION  MOTION2[];

NJS_MOTION_LINK  motionlink;
NJS_ACTION_LINK  actionlink;

.....
actionlink.motionlink = &motionlink;
actionlink.object = OBJECT;
motionlink.motion[0] = MOTION1; /* link source */
motionlink.motion[1] = MOTION2; /* link destination */
/* interpolate 1.5 frames of MOTION2 from the final frame of MOTION1 */
motionlink.frame[0] = MOTION1->nbFrame-1;
motionlink.frame[1] = 1.5;

.....
njFastActionLink( &actionlink, ff/STEPS ); /* Specify from 0 - 1 for frames */

ff+=0.2f;
if( ff >= STEPS - 1 ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion has been added. In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njFastDrawObject, so the light source is bright light.

## RELATED TOPICS

Related Items

[njActionLink\(\)](#)

[njDrawMotionLink\(\)](#)

[njFastDrawMotionLink\(\)](#)

# njFastDrawMotion

Draws Motion.

## FORMAT

```
#include <Ninja.h>
void njFastDrawMotion( *object, *motion, frame )
NJS_OBJECT  *object
NJS_MOTION  *motion
Float       frame
```

## PARAMETERS

### **object**

pointer for the object structure

### **motion**

pointer for the motion structure

### **frame**

frame number

## RETURN VALUE

None

## ERROR VALUE

None

## FUNCTION

- This function draws motion according to the motion data.
- Because floating-point data can be used for frame numbers, the motion data can be interpolated.
- In the case of a motion consisting of 10 frames, the frame numbers can range from 0 to just under 10 (i.e., 9.9999).
- For example, if the frame number is 0.5, the values used for drawing are interpolated as exactly halfway between the values for frame 0 and the values for frame 1.
- Interpolation from 9 to just under 10 (9.9999) becomes interpolation between the last frame and the first frame.

## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample
#define MOTION    motion_sample
extern NJS_MOTION MOTION[];
extern NJS_OBJECT OBJECT[];

.....
njFastDrawMotion( OBJECT,MOTION,ff );

ff+=0.2f;
if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
.....
```

## NOTES

- This function is identical to njAction(), except that the action structure has been divided into objects and motion. This function corresponds to the model drawing function njFastDrawObject, so the light source is bright light.

## RELATED TOPICS

[njFastAction\(\)](#)  
[njAction\(\)](#)  
[njDrawMotion\(\)](#)

# **njFastDrawMotionLink**

**Links motions.**

---

## **FORMAT**

```
#include <Ninja.h>
void njFastDrawMotionLink( object, motionlink, frame )
NJS_OBJECT      *object
NJS_MOTION_LINK *motionlink
Float           frame
```

## **PARAMETERS**

### **object**

Pointer to object structure

### **motionlink**

Pointer to motion link structure

### **frame**

Frame number (from 0 to 1)

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**



## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample
#define MOTION1    motion_sample1 /* described in nam */
#define MOTION2    motion_sample2 /* described in nam */
#define STEPS      60

extern NJS_MOTION  MOTION1[];
extern NJS_MOTION  MOTION2[];

NJS_MOTION_LINK  motionlink;

.....
motionlink.motion[0] = MOTION1; /* link source */
motionlink.motion[1] = MOTION2; /* link destination */
/* interpolate 1.5 frames of MOTION2 from the final frame of MOTION1 */
otionlink.frame[0] = MOTION1->nbFrame-1;
    motionlink.frame[1] = 1.5;

.....
njFastDrawMotionLink( OBJECT, &motionlink, ff/STEPS );
/* Specify from 0 - 1 for frames */

ff+=0.2f;
if( ff >= STEPS - 1 ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion has been added. In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njFastDrawObject, so the light source is bright light.

## RELATED TOPICS

Related Items

[njActionLink\(\)](#)

[njDrawMotionLink\(\)](#)

[njFastActionLink\(\)](#)

## **njFastDrawShapeMotion**      Executes motion that includes shapes.

---

### **FORMAT**

```
#include <Ninja.h>
void  njFastDrawMotion( object, motion, frame )
NJS_OBJECT  *object
NJS_MOTION  *motion
NJS_MOTION  *shape
Float       frame
```

### **PARAMETERS**

#### **object**

Pointer to object structure

#### **motion**

Pointer to motion structure

#### **shape**

Pointer to motion structure (vertex animation data)

#### **frame**

Frame number

### **RETURN VALUE**

None

### **ERROR VALUE**

None

## FUNCTION

- This function draws motion according to the motion data.
- Because floating-point data can be used for frame numbers, the motion data can be interpolated.
- In the case of a motion consisting of 10 frames, the frame numbers can range from 0 to just under 10 (i.e., 9.9999).
- For example, if the frame number is 0.5, the values used for drawing are interpolated as exactly halfway between the values for frame 0 and the values for frame 1.
- Interpolation from 9 to just under 10 (9.9999) becomes interpolation between the last frame and the first frame.
- This also applies to vertex animation.

## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample /* described within NJA (NAM) file */
#define MOTION    motion_sample /* described within NAM file */
#define SHAPE     shape_sample  /* described within NAS file */
extern NJS_OBJECT OBJECT[];
extern NJS_MOTION MOTION[];
extern NJS_MOTION SHAPE[];
float  buf[10000];
float  ff=0.f;

.....
InitShape(buf);

.....
njFastDrawShapeMotion( OBJECT,MOTION,SHAPE,ff );

ff+=0.2f;
if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion and shapes (vertex animation) have been added.

In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njFastDrawObject, so the light source is bright light.

## RELATED TOPICS

`njInitShape()`

`njDrawShapeMotion()`

## **njFastDrawShapeMotionLink**      Links motions that include shapes.

---

### **FORMAT**

```
#include <Ninja.h>
void njFastDrawShapeMotionLink( object, motionlink, shapelink, frame )
NJS_OBJECT      *object
NJS_MOTION_LINK *motionlink
NJS_MOTION_LINK *shapelink
Float           frame
```

### **PARAMETERS**

#### **object**

Pointer to object structure

#### **motionlink**

Pointer to motion link structure

#### **shapelink**

Pointer to motion link structure (vertex animation)

#### **frame**

Frame number (from 0 to 1)

### **RETURN VALUE**

None

### **ERROR VALUE**

None

### **FUNCTION**

- This function links two motion structures.
- The frames in the motion link structure are the link source and link destination motion frames.
- When the frame parameter is 0, the frame is part of the motion of the link source frame; when the frame parameter is 1, the frame is part of the motion of the link destination frame.
- Other values ( $0 < ff < 1$ ) interpolate the motion between the link destination frame and the link source frame.
- This also applies to shapes (vertex animation).

## EXAMPLE

```
#include <NINJA.H>

#define OBJECT    object_sample
#define MOTION1    motion_sample1 /* described within NAM file */
#define MOTION2    motion_sample2 /* described within NAM file */
#define SHAPE1    shape_sample1  /* described within NAS file */
#define SHAPE2    shape_sample2  /* described within NAS file */
#define STEPS      60

extern NJS_MOTION MOTION1[];
extern NJS_MOTION MOTION2[];
extern NJS_MOTION SHAPE1[];
extern NJS_MOTION SHAPE2[];

NJS_MOTION_LINK motionlink;
NJS_MOTION_LINK shapelink;

.....
motionlink.motion[0] = MOTION1; /* link source */
motionlink.motion[1] = MOTION2; /* link destination */
shapelink.motion[0] = SHAPE1; /* link source */
shapelink.motion[1] = SHAPE2; /* link destination */
/* interpolate 1.5 frames of MOTION2 from the final frame of MOTION1 */
motionlink.frame[0] =
shapelink.frame[0] = MOTION1->nbFrame-1;
motionlink.frame[1] =
shapelink.frame[1] = 1.5;

.....
njFastDrawShapeMotionLink( OBJECT, &motionlink, &shapelink, ff/STEPS );
/* Specify from 0 - 1 for frames */

ff+=0.2f;
if( ff >= STEPS - 1 ) ff=0.f;
.....
```

## NOTES

The drawing performed by this function is identical to model drawing, except that motion and shapes (vertex animation) have been added.

In other words, all of the njControl3D settings are identical. This function corresponds to the model drawing function njFastDrawObject, so the light source is bright light.

## RELATED TOPICS

`njInitShape()`

`njDrawShapeMotionLink()`

## **njGetMotionRotate**

This is a node motion information function (rotate).

---

### **Syntax**

```
#include <Ninja.h>
void  njGetMotionRotate( Float *rot, Int spline_flag );
Float *rot
Int    spline_flag
```

### **Arguments**

**rot**

Information on rotate (but rot[3])

**spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

Information on rotate at each node is stored in the argument.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION    motion_sample
#define MDATA3    ((NJS_MDATA3 *) MOTION->mdata)
#define LINEAR    0
#define SPLINE    1
extern NJS_MOTION MOTION[ ];
extern NJS_OBJECT OBJECT[ ];

Float ff = 0.f;

Int spline_flg = SPLINE;

/* This mdata is for getting object information */
NJS_MDATA3 *mdata;

.....

Sint32 njUserMain(void)
{
    .....

    mdata = MDATA3;
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;

    .....
}

PushPopMotion(NJS_OBJECT *obj)
{
    float pos[3], scl[3];
    Angle ang[3];
    njPushMatrix( NULL );
    /* Execute motion function */
    /* Same as processing of the following function */
    /* njMotionTranslate( NULL, obj, spline_flg ); */
    /* njMotionRotateXYZ( NULL, obj, spline_flg ); */
    /* njMotionScale( NULL, obj, spline_flg ); */
    /* njSetNextMotionNode(); */
    if (mdata->p[0] != NULL) {
        njGetMotionTranslate( pos, spline_flg );
    } else {
```

```
pos[0] = obj->pos[0];
pos[1] = obj->pos[1];
pos[2] = obj->pos[2];
}
if (mdata->p[1] != NULL) {
    njGetMotionRotate( ang, spline_flg );
} else {
    ang[0] = obj->ang[0];
    ang[1] = obj->ang[1];
    ang[2] = obj->ang[2];
}
if (mdata->p[2] != NULL) {
    njGetMotionScale( scl, spline_flg );
} else {
    scl[0] = obj->scl[0];
    scl[1] = obj->scl[1];
    scl[2] = obj->scl[2];
}
njTranslate( NULL, pos[0], pos[1], pos[2] );
njRotateXYZ( NULL, ang[0], ang[1], ang[2] );
njScale( NULL, scl[0], scl[1], scl[2] );
njSetNextMotionNode();
mdata++;
/* Draw picture of model */
if( obj->model != NULL )
    njDrawModel( obj->model );
if( obj->child != NULL )
    PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
if( obj->sibling != NULL )
    PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
```

.....



## Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

## Related Items

- `njSetCurrentMotion`
- `njMotionTransform`
- `njMotionTransformZXY`
- `njMotionTranslate`
- `njMotionRotateXYZ`
- `njMotionRotateZXY`
- `njMotionScale`
- `njGetMotionTranslate`
- `njGetMotionScale`
- `njSetNextMotionNode`

## **njGetMotionScale**

This is a node motion info mation function (scale).

---

### **Syntax**

```
#include <Ninja.h>
void    njGetMotionScale( Float *scl, Int spline_flag );
Float   *scl
Int      spline_flag
```

### **Arguments**

**scl**

Information on scale (but scl[3])

**spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

Information on scale at each node is stored in the argument.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)

### **Usage Example**

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION    motion_sample
#define MDATA3    ((NJS_MDATA3 *) MOTION->mdata)
#define LINEAR    0
#define SPLINE    1
extern NJS_MOTION MOTION[ ];
extern NJS_OBJECT OBJECT[ ];
```

```
Float ff = 0.f;
Int spline_flg = SPLINE;
/* This mdata is for getting object information */
NJS_MDATA3 *mdata;

.....
Sint32 njUserMain(void)
{
    .....
    mdata = MDATA3;
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    float pos[3], scl[3];
    Angle ang[3];
    njPushMatrix( NULL );
    /* Execute motion function */
    /* Same as processing of the following function */
    /* njMotionTranslate( NULL, obj, spline_flg ); */
    /* njMotionRotateXYZ( NULL, obj, spline_flg ); */
    /* njMotionScale( NULL, obj, spline_flg ); */
    /* njSetNextMotionNode(); */
    if (mdata->p[0] != NULL) {
        njGetMotionTranslate( pos, spline_flg );
    } else {
        pos[0] = obj->pos[0];
        pos[1] = obj->pos[1];
        pos[2] = obj->pos[2];
    }
    if (mdata->p[1] != NULL) {
        njGetMotionRotate( ang, spline_flg );
    } else {
        ang[0] = obj->ang[0];
        ang[1] = obj->ang[1];
        ang[2] = obj->ang[2];
    }
    if (mdata->p[2] != NULL) {
        njGetMotionScale( scl, spline_flg );
    } else {
        scl[0] = obj->scl[0];
        scl[1] = obj->scl[1];
        scl[2] = obj->scl[2];
    }
    njTranslate( NULL, pos[0], pos[1], pos[2] );
    njRotateXYZ( NULL, ang[0], ang[1], ang[2] );
    njScale( NULL, scl[0], scl[1], scl[2] );
}
```

```
njSetNextMotionNode();
mdata++;
/* Draw picture of model */
if( obj->model != NULL )
njDrawModel( obj->model );
if( obj->child != NULL )
PushPopMotion( (NJS_OBJECT*)obj->child );

njPopMatrix( 1 );
if( obj->sibling != NULL )
PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
.....
Remark
This function is used as one of the functions for navigating the parent-child hierarchy
(push/pop).
Related Items
njSetCurrentMotion
njMotionTransform
njMotionTransformZXY
njMotionTranslate
njMotionRotateXYZ
njMotionRotateZXY
njMotionScale
njGetMotionTranslate
njGetMotionRotate
njSetNextMotionNode
```

## njGetMotionTranslate

This is a node motion information function (move).

---

### Syntax

```
#include <Ninja.h>
void   njGetMotionTranslate( Float *pos, Int spline_flag );
Float  *pos
Int     spline_flag
```

### Arguments

#### **pos**

Information on move (but pos[3])

#### **spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### Return Value

None

### Error

None

### Description

Information on move at each node is stored in the argument.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION    motion_sample
#define MDATA3    ((NJS_MDATA3 *) MOTION->mdata)
#define LINEAR    0
#define SPLINE    1
extern NJS_MOTION MOTION[];
extern NJS_OBJECT OBJECT[];
Float  ff = 0.f;
Int    spline_flg = SPLINE;
/* This mdata is for getting object information */
NJS_MDATA3 *mdata;
    .....
Sint32 njUserMain(void)
{
    .....
mdata = MDATA3;
njSetCurrentMotion(MOTION, ff );
PushPopMotion( OBJECT );
ff+=0.2f;
if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
float pos[3], scl[3];
Angle ang[3];
njPushMatrix( NULL );
/* Execute motion function */
/* Same as processing of the following function */
/* njMotionTranslate( NULL, obj, spline_flg ); */
/* njMotionRotateXYZ( NULL, obj, spline_flg ); */
/* njMotionScale( NULL, obj, spline_flg ); */
/* njSetNextMotionNode(); */
if (mdata->p[0] != NULL) {
njGetMotionTranslate( pos, spline_flg );
} else {
pos[0] = obj->pos[0];
pos[1] = obj->pos[1];
pos[2] = obj->pos[2];
}
if (mdata->p[1] != NULL) {
njGetMotionRotate( ang, spline_flg );
} else {
ang[0] = obj->ang[0];
ang[1] = obj->ang[1];
ang[2] = obj->ang[2];
}
}
```

```
if (mdata->p[2] != NULL) {
  njGetMotionScale( scl, spline_flg );
} else {
  scl[0] = obj->scl[0];
  scl[1] = obj->scl[1];
  scl[2] = obj->scl[2];
}
njTranslate( NULL, pos[0], pos[1], pos[2] );
njRotateXYZ( NULL, ang[0], ang[1], ang[2] );
njScale( NULL, scl[0], scl[1], scl[2] );
njSetNextMotionNode();
mdata++;
/* Draw picture of model */
if( obj->model != NULL )
  njDrawModel( obj->model );
if( obj->child != NULL )
  PushPopMotion( (NJS_OBJECT*)obj->child );
njPopMatrix( 1 );
if( obj->sibling != NULL )
  PushPopMotion( (NJS_OBJECT*)obj->sibling );
}

.....
```

## Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

## Related Items

```
njSetCurrentMotion
njMotionTransform
njMotionTransformZXY
njMotionTranslate
njMotionRotateXYZ
njMotionRotateZXY
njMotionScale
njGetMotionRotate
njGetMotionScale
njSetNextMotionNode
```

## **njMotionRotateXYZ**    This is a node motion function (rotate).

---

### **Syntax**

```
#include <Ninja.h>
void  njMotionRotateXYZ( NJS_MATRIX *m, NJS_OBJECT *obj, Int spline_flag );
NJS_MOTION  *motion
NJS_OBJECT  *obj
Int          spline_flag
```

### **Arguments**

**motion**

Pointer to motion structure

**obj**

Pointer to object structure

**spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

This function reflects the matrix obtained from rotate motion at each node.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)



## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION    motion_sample
#define LINEAR    0
#define SPLINE    1
extern NJS_MOTION MOTION[];
extern NJS_OBJECT OBJECT[];
Float ff = 0.f;
Int spline_flg = SPLINE;

.....
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTranslate( NULL, obj, spline_flg );
    njMotionRotateXYZ( NULL, obj, spline_flg );
    njMotionScale( NULL, obj, spline_flg );
    njSetNextMotionNode();
    /* Draw picture of model */
    if( obj->model != NULL )
        njDrawModel( obj->model );
    if( obj->child != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
.....
```

### Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).  
These functions (motion of each node) are similar, with exception of the 3D control,  
to the njAction and njDrawMotion groups.

### Related Items

- `njSetCurrentMotion`
- `njMotionTransform`
- `njMotionTransformZXY`
- `njMotionTranslate`
- `njMotionRotateZXY`
- `njMotionScale`
- `njGetMotionTranslate`
- `njGetMotionRotate`
- `njGetMotionScale`
- `njSetNextMotionNode`

## **njMotionRotateZX** This is a node motion function (ZXY rotate).

---

### **Syntax**

```
#include <Ninja.h>
void      njMotionRotateZXY( NJS_MATRIX *m, NJS_OBJECT *obj, Int spline_flag );
NJS_MOTION *motion
NJS_OBJECT *obj
Int      spline_flag
```

### **Arguments**

**motion**

Pointer to motion structure

**obj**

Pointer to object structure

**spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

This function reflects the matrix obtained from rotate (ZXY type) motion at each node.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION     motion_sample
#define LINEAR     0
#define SPLINE     1
extern NJS_MOTION  MOTION[];
extern NJS_OBJECT  OBJECT[];
Float  ff = 0.f;
Int  spline_flg = SPLINE;
.....
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTranslate( NULL, obj, spline_flg );
    njMotionRotateZXY( NULL, obj, spline_flg );
    njMotionScale( NULL, obj, spline_flg );
    njSetNextMotionNode();
    /* Draw picture of model */
    if( obj->model != NULL )
        njDrawModel( obj->model );
    if( obj->child != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
.....
```

## Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

## Related Items

- `njSetCurrentMotion`
- `njMotionTransform`
- `njMotionTransformZXY`
- `njMotionTranslate`
- `njMotionRotateXYZ`
- `njMotionScale`
- `njGetMotionTranslate`
- `njGetMotionRotate`
- `njGetMotionScale`
- `njSetNextMotionNode`

## **njMotionScale**

This is a node motion function (scale).

---

### **Syntax**

```
#include <Ninja.h>
void  njMotionScale( NJS_MATRIX *m, NJS_OBJECT *obj, Int spline_flag );
NJS_MOTION  *motion
NJS_OBJECT  *obj
Int          spline_flag
```

### **Arguments**

#### **motion**

Pointer to motion structure

#### **obj**

Pointer to object structure

#### **spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

This function reflects the matrix obtained from scale motion at each node.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION     motion_sample
#define LINEAR     0
#define SPLINE     1
extern NJS_MOTION  MOTION[];
extern NJS_OBJECT  OBJECT[];
Float  ff = 0.f;
Int    spline_flg = SPLINE;

.....
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTranslate( NULL, obj, spline_flg );
    njMotionRotateXYZ( NULL, obj, spline_flg );
    njMotionScale( NULL, obj, spline_flg );
    njSetNextMotionNode();
    /* Draw picture of model */
    if( obj->model != NULL )
        njDrawModel( obj->model );
    if( obj->child != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
.....
```

### Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

### Related Items

- `njSetCurrentMotion`
- `njMotionTransform`
- `njMotionTransformZXY`
- `njMotionTranslate`
- `njMotionRotateXYZ`
- `njMotionRotateZXY`
- `njGetMotionTranslate`
- `njGetMotionRotate`
- `njGetMotionScale`
- `njSetNextMotionNode`



# njMotionTransform

# This is a node motion function

---

## Syntax

```
#include <Ninja.h>
void njMotionTransform( NJS_MATRIX *m, NJS_OBJECT *obj, Int spline_flag );
NJS_MOTION  *motion
NJS_OBJECT  *obj
Int          spline_flag
```

## Arguments

## Return Value

None

## Error

None

## Description

This function reflects the matrix obtained from motion (move, rotate, scale) at each node.

To obtain information such as frame motion,

this function must call the njSetCurrentMotion function before use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION    motion_sample
#define LINEAR    0
#define SPLINE    1
extern NJS_MOTION MOTION[ ];
extern NJS_OBJECT OBJECT[ ];
Float  ff = 0.f;
Int    spline_flg = SPLINE;
.....
```

```
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTransform( NULL, obj, spline_flg );
    /* Draw picture of model */
    if( obj->model != NULL )
        njDrawModel( obj->model );
    if( obj->child != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
.....
```

### Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

### Related Items

```
njSetCurrentMotion
njMotionTransformZXY
njMotionTranslate
njMotionRotateXYZ
njMotionRotateZXY
njMotionScale
njGetMotionTranslate
njGetMotionRotate
njGetMotionScale
njSetNextMotionNode
```

## **njMotionTransformZXY**

This is a node motion function (ZXY rotate).

---

### **Syntax**

```
#include <Ninja.h>
void    njMotionTransformZXY( NJS_MATRIX *m, NJS_OBJECT *obj, Int spline_flag );
NJS_MOTION  *motion
NJS_OBJECT  *obj
Int          spline_flag
```

### **Arguments**

#### **motion**

Pointer to motion structure

#### **obj**

Pointer to object structure

#### **spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

This function reflects the matrix obtained from motion (move, rotate, scale) at each node.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION     motion_sample
#define LINEAR     0
#define SPLINE     1
extern NJS_MOTION  MOTION[];
extern NJS_OBJECT  OBJECT[];
Float  ff = 0.f;
Int  spline_flg = SPLINE;

    .....
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTransformZXY( NULL, obj, spline_flg );
    /* Draw picture of model */
    if( obj->model != NULL )
        njDrawModel( obj->model );
    if( obj->child != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->sibling );
}

    .....
```

## Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop). These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

## Related Items

- `njSetCurrentMotion`
- `njMotionTransform`
- `njMotionTranslate`
- `njMotionRotateXYZ`
- `njMotionRotateZXY`
- `njMotionScale`
- `njGetMotionTranslate`
- `njGetMotionRotate`
- `njGetMotionScale`
- `njSetNextMotionNode`

## **njMotionTranslate**      This is a node motion function (move).

---

### **Syntax**

```
#include <Ninja.h>
void  njMotionTranslate( NJS_MATRIX *m, NJS_OBJECT *obj, Int spline_flag );
NJS_MOTION  *motion
NJS_OBJECT  *obj
Int          spline_flag
```

### **Arguments**

**motion**

Pointer to motion structure

**obj**

Pointer to object structure

**spline\_flag**

Spline attribute flag (0: linear and 1:spline)

### **Return Value**

None

### **Error**

None

### **Description**

This function reflects the matrix obtained from move motion at each node.

To obtain information such as frame motion, this function must call the njSetCurrentMotion function before use, and, if necessary (if within the hierarchical structure), must call njSetNextMotionNode to update the node information after use. (See example.)

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION     motion_sample
#define LINEAR     0
#define SPLINE     1
extern NJS_MOTION  MOTION[];
extern NJS_OBJECT  OBJECT[];
Float  ff = 0.f;
Int    spline_flg = SPLINE;

    .....
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTranslate( NULL, obj, spline_flg );
    njMotionRotateXYZ( NULL, obj, spline_flg );
    njMotionScale( NULL, obj, spline_flg );
    njSetNextMotionNode();
    /* Draw picture of model */
    if( obj->model != NULL )
        njDrawModel( obj->model );
    if( obj->child != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
        PushPopMotion( (NJS_OBJECT*)obj->sibling );
}

    .....
```

### Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

### Related Items

- `njSetCurrentMotion`
- `njMotionTransform`
- `njMotionTransformZXY`
- `njMotionRotateXYZ`
- `njMotionRotateZXY`
- `njMotionScale`
- `njGetMotionTranslate`
- `njGetMotionRotate`
- `njGetMotionScale`
- `njSetNextMotionNode`



## njSetCurrentMotion

Set parameters of the node motion function.

---

### Syntax

```
#include <Ninja.h>
void njSetCurrentMotion(NJS_MOTION *motion, Float frame );
NJS_MOTION  *motion
Float       frame
```

### Arguments

**motion**

Pointer to motion structure

**frame**

Frame number

### Return Value

None

### Error

None

### Description

Set motion parameters at each node.

## Usage Example

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION    motion_sample
#define LINEAR    0
#define SPLINE    1
extern NJS_MOTION MOTION[];
extern NJS_OBJECT OBJECT[];
Float  ff = 0.f;
Int    spline_flg = SPLINE;

    .....
Sint32 njUserMain(void)
{
    .....
    njSetCurrentMotion(MOTION, ff );
    PushPopMotion( OBJECT );
    ff+=0.2f;
    if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
    .....
}
PushPopMotion(NJS_OBJECT *obj)
{
    njPushMatrix( NULL );
    /* Execute motion function */
    njMotionTranslate( NULL, obj, spline_flg );
    njMotionRotateXYZ( NULL, obj, spline_flg );
    njMotionScale( NULL, obj, spline_flg );
    njSetNextMotionNode();
    /* Draw picture of model */
    if( obj->model != NULL )
    njDrawModel( obj->model );
    if( obj->child != NULL )
    PushPopMotion( (NJS_OBJECT*)obj->child );
    njPopMatrix( 1 );
    if( obj->sibling != NULL )
    PushPopMotion( (NJS_OBJECT*)obj->sibling );
}

    .....
```

## Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

## Related Items

- `njMotionTransform`
- `njMotionTransformZXY`
- `njMotionTranslate`
- `njMotionRotateXYZ`
- `njMotionRotateZXY`
- `njMotionScale`
- `njGetMotionTranslate`
- `njGetMotionRotate`
- `njGetMotionScale`
- `njSetNextMotionNode`

## **njSetNextMotionNode**

Update node of node motion function.

---

### **Syntax**

```
#include <Ninja.h>
void njSetNextMotionNode( void );
```

### **Arguments**

None

### **Return Value**

None

### **Error**

None

### **Description**

Update node.

Please do not perform setting at njMotionTransform and njMotionTransformZXY.

### **Usage Example**

```
#include <NINJA.H>
#define OBJECT    object_sample
#define MOTION     motion_sample
#define LINEAR     0
#define SPLINE     1
extern NJS_MOTION MOTION[];
extern NJS_OBJECT OBJECT[];
Float  ff = 0.f;
Int    spline_flg = SPLINE;
.....
Sint32 njUserMain(void)
{
```

```
.....
njSetCurrentMotion(MOTION, ff );
PushPopMotion( OBJECT );
ff+=0.2f;
if( ff >= (MOTION->nbFrame-1) ) ff=0.f;
.....
}
PushPopMotion(NJS_OBJECT *obj)
{
  njPushMatrix( NULL );
  /* Execute motion function */
  njMotionTranslate( NULL, obj, spline_flg );
  njMotionRotateXYZ( NULL, obj, spline_flg );
  njMotionScale( NULL, obj, spline_flg );
  njSetNextMotionNode();
  /* Draw picture of model */
  if( obj->model != NULL )
    njDrawModel( obj->model );
  if( obj->child != NULL )
    PushPopMotion( (NJS_OBJECT*)obj->child );
  njPopMatrix( 1 );
  if( obj->sibling != NULL )
    PushPopMotion( (NJS_OBJECT*)obj->sibling );
}
.....
```

## Remark

This function is used as one of the functions for navigating the parent-child hierarchy (push/pop).

These functions (motion of each node) are similar, with exception of the 3D control, to the njAction and njDrawMotion groups.

## Related Items

```
njSetCurrentMotion
njMotionTransform
njMotionTransformZXY
njMotionTranslate
njMotionRotateXYZ
njMotionRotateZXY
njMotionScale
njGetMotionTranslate
njGetMotionRotate
njGetMotionScale
```

# **njSetMotionCallback**

Registering motion callback routine.

---

## **FORMAT**

```
#include <Ninja.h>
void njSetMotionCallback( func )
void (*func)(NJS_OBJECT *obj)
```

## **PARAMETERS**

**func**

callback routine pointer

**obj**

callback parameters

## **RETURN VALUE**

None

## **ERROR VALUE**

None

## **FUNCTION**

- Registers motion callback routine.
- After motion matrix operation, it is called right before drawing.

## **EXAMPLE**

## **NOTES**

## **RELATED TOPICS**

`njInit3D()`

`njDrawModel()`

`njDrawObject()`

`njControl3D()`

`njAction()`



## *16. Memory Functions*

---

### **Contents**

njMemCopy.....	Memory Copy. ....	NLS-502
njMemCopy4.....	Memory Copy. (Word) .....	NLS-503
njMemCopy2.....	Memory Copy. (Long) .....	NLS-504

# njMemCopy

Memory Copy.

## FORMAT

```
#include <Ninja.h>
void njMemCopy( *dst, *src, num)
void    *dst
void    *src
Sint32  num
```

## PARAMETERS

**\*dst**

Destination

**\*src**

Source

**num**

Number of bytes

## RETURN VALUE

None

## FUNCTION

- Copies num bytes from src to dst.

## EXAMPLE

```
Uint8 dst[256];
Uint8 src[256];

/* Copies 256 bytes */
njMemCopy(dst,src,256);
```

## NOTES

## RELATED TOPICS

[njMemCopy2](#)

[njMemCopy4](#)



# njMemCopy4

Memory Copy. (Word)

---

## FORMAT

```
#include <Ninja.h>
void njMemCopy2( *dst, *src, num)
void                                     *dst
void                                     *src
Sint32                                    num
```

## PARAMETERS

**\*dst**

Destination

**\*src**

Source

**num**

Number of words

## RETURN VALUE

None

## FUNCTION

- Copies num words from src to dst.

## EXAMPLE

```
Uint16 dst[256];
Uint16 src[256];

/* Copies 256 words */
njMemCopy2(dst,src,256);
```

## NOTES

## RELATED TOPICS

njMemCopy

njMemCopy4

# njMemCopy2

Memory Copy. (Long)

---

## FORMAT

```
#include <Ninja.h>
void njMemCopy4( *dst, *src, num)
void    *dst
void    *src
Sint32  num
```

## PARAMETERS

**\*dst**

Destination

**\*src**

Source

**num**

Number of long

## RETURN VALUE

None

## FUNCTION

- Copies num long from src to dst.

## EXAMPLE

```
Uint32 dst[256];
Uint32 src[256];

/* Copies 256 long */
njMemCopy4(dst,src,256);
```

## NOTES

## RELATED TOPICS

[njMemCopy\(\)](#)

[njMemCopy2\(\)](#)



# *17. Drawing Functions*

---

## **Contents**

njDrawPolygon.....	Draws polygons without textures. ....	NLS-506
njDrawTexture.....	Draw texture polygons. ....	NLS-508

## njDrawPolygon

Draws polygons without textures.

---

### FORMAT

```
#include <Ninja.h>
void njDrawPolygon( *polygon, count, trans )
NJS_POLYGON_VTX  *polygon
Int      count
Int      trans
```

### PARAMETERS

#### **\*polygon**

Pointer for NJS\_POLYGON\_VTX structure

#### **count**

Number of vertices

#### **trans**

Translucent valid(TRUE), invalid(FALSE)

### RETURN VALUE

None

### FUNCTION

- Draws polygons without texture. Count vertices in strip format, and set the number to count.
- Set trans to TRUE when translucent is used and to FALSE when it is not used.

## EXAMPLE

Draws rectangle.

Vertices order as follows:

```
1  3  
2  4
```

```
NJS_POLYGON_VTX  poly[4];
```

```
poly[0].x = 100.f;  
poly[0].y = 100.f;  
poly[0].z = 0.5f;  
poly[0].col = 0xFFFFFFFF;
```

```
poly[1].x = 100.f;  
poly[1].y = 300.f;  
poly[1].z = 0.5f;  
poly[1].col = 0xFFFFFFFF;
```

```
poly[2].x = 300.f;  
poly[2].y = 100.f;  
poly[2].z = 0.5f;  
poly[2].col = 0xFFFFFFFF;
```

```
poly[3].x = 300.f;  
poly[3].y = 300.f;  
poly[3].z = 0.5f;  
poly[3].col = 0xFFFFFFFF;
```

```
njDrawPolygon( poly, 4, FALSE);
```

## NOTES

## RELATED TOPICS

[njDrawTexture](#)

# njDrawTexture

Draw texture polygons.

---

## FORMAT

```
#include <Ninja.h>
void njDrawTexture( *polygon, count, tex, trans )
NJS_POLYGON_VTX  *polygon
Int      count
Int      tex
Int      trans
```

## PARAMETERS

### **\*polygon**

Pointer for NJS\_POLYGON\_VTX structure

### **count**

Number of vertices

### **tex**

Texture global index

### **trans**

Translucent valid(TRUE), invalid(FALSE)

## RETURN VALUE

None

## FUNCTION

- Draws texture polygons. Count vertices in strip format, and set the number to count.
- Before using this function, it is necessary to initialize, load, and set textures.
- Set trans to TRUE when translucent is used and to FALSE when it is not used.

## EXAMPLE

Draw rectangles.

Vertices order as follows:

```
1 3
2 4
```

```
NJS_TEXNAME texname[1];
NJS_TEXLIST texlist = {texname,1};
NJS_TEXMEMLIST tex[1];

NJS_TEXTURE_VTX poly[4];

/* Initializing textures*/
njInitTexture( tex,1);
njSetTextureName(&texname[0],"file0.pvr",0,NJD_TEXATTR_TYPE_FILE|
                                                         NJD_TEXATTR_GLOBALINDEX);

njLoadTexture( &texlist );
njSetTexture( &texlist );

poly[0].x = 100.f;
poly[0].y = 100.f;
poly[0].z = 0.5f;
poly[0].u = 0.f;
poly[0].v = 0.f;
poly[0].col = 0xFFFFFFFF;

poly[1].x = 100.f;
poly[1].y = 300.f;
poly[1].z = 0.5f;
poly[1].u = 0.f;
poly[1].v = 1.f;
poly[1].col = 0xFFFFFFFF;

poly[2].x = 300.f;
poly[2].y = 100.f;
poly[2].z = 0.5f;
poly[2].u = 1.f;
poly[2].v = 0.f;
poly[2].col = 0xFFFFFFFF;

poly[3].x = 300.f;
poly[3].y = 300.f;
poly[3].z = 0.5f;
poly[3].u = 1.f;
poly[3].v = 1.f;
poly[3].col = 0xFFFFFFFF;

/* Map the texture of global index #0 to polygon */
njDrawTexture( poly, 4, 0, FALSE);
```

## NOTES

## RELATED TOPICS

`njDrawPolygon`







## *18. Input Functions*

---

### **Contents**

njGetPeripheral .....	Gets information of input device. (Peripheral).	.....	NLS-512
njPrintPeripheralInfo .....	Displays peripheral condition.	.....	NLS-516

## njGetPeripheral

Gets information of input device. (Peripheral).

---

### FORMAT

```
#include <NinjaWin.h>
NJS_PERIPHERAL *njGetPeripheral(port)
Uint32 port
```

### PARAMETER

**port**

Peripheral port number

NJD_PORT_SYSKEYBOARD	System keyboard
NJD_PORT_SYSMOUSE	System mouse
NJD_PORT_JOYSTICK1	Joystick port 1
:	
NJD_PORT_JOYSTICK16	Joystick port 16

### RETURN VALUE

The address of peripheral structure

### FUNCTION

- Gets information about peripheral connection, kinds, and button.
- By referring members of peripheral structure, applications can get various information.
- Explanation of NJS\_PERIPHERAL structure members

The kind of connected peripheral

NJD_DEV_NODEVICE	Disconnected
NJD_DEV_SYSKEYBOARD	System keyboard
NJD_DEV_SYSMOUSE	System mouse
NJD_DEV_JOYSTICK	Joystick
NJD_DEV_UNDEFINED	Undefined

---

 **NOTE:** Do not mix them up with peripheral port numbers.

---

**on button information(ON)**

Button information is stored.

If buttons are pushed, the bit which matches the button becomes 1, if not pushed, the bit becomes 0.

Gets ON/OFF of buttons using AND with the following macro.

NJD_DGT_KU	Directional key Up
NJD_DGT_KD	Directional key Down
NJD_DGT_KL	Directional key Left
NJD_DGT_KR	Directional key Right
NJD_DGT_TA	A button
NJD_DGT_TB	B button
NJD_DGT_TC	C button
NJD_DGT_TX	X button
NJD_DGT_TY	Y button
NJD_DGT_TZ	Z button
NJD_DGT_TL	L button
NJD_DGT_TR	R button
NJD_DGT_ST	START button

**off button information(OFF)**

Contrary to member on, the bit which matches the button becomes 0, if not pushed, the bit becomes 1.

**push button information(ON edge)**

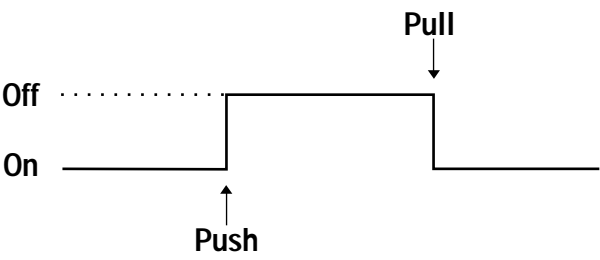
The moment the button is pushed, the bit becomes 1.

Uses it for edge detection.

**pull button information(OFF edge)**

The moment the button is pulled, the bit becomes 1.

Uses it for edge detection.



**x,y,z,r,u,v**

Gets data of analog axis.

The value must be 16bit and from -32768 to 32767.

**extend**

The address of extended data corresponding to the kind of peripherals is set.

Not available!!

**name**

The pointer of peripheral name string (ASCIIZ) is set.

- Keyboard and digital pad

Digital Pad	Keyboard
Direction key UP	Cursol UP or Ten key 8
Direction key DOWN	Cursol DOWN or Ten key 2
Direction key LEFT	Cursol LEFT or Ten key 4
Direction key RIGHT	Cursol RIGHT or Ten key 6
A Button	Z Key
B Button	X Key
C Button	C Key
X Button	A Key
Y Button	S Key
Z Button	D Key
L Button	Q Key
R Button	E Key

START Button	SPACE Key or RETURN Key
--------------	-------------------------

## EXAMPLE

```
Sint32 njUserMain(void)
{
    NJS_PERIPHERAL* per;

    /* Gets information of peripheral which is
     * connected to joystick port1.
     */
    per = njGetPeripheral(NJD_DEV_JOYSTICK1);

    if(per->id == NJD_DEV_NODEVICE){
        /* Peripheral disconnected */
    }

    if(per->push & NJD_DGT_ST){
        /* Start button is pushed (edge) */
    }

    if(per->on & NJD_DGT_KU){
        /* Directional key is pushed */
    }

    if((per->on & (NJD_DGT_TA | NJD_DGT_TB))==(NJD_DGT_TA | NJD_DGT_TB)){
        /* Judgement for some buttons pushed
         * simultaneously - Both button A and B are pushed.
         */
    }

    return NJS_USER_CONTINUE;
}
```

## NOTES

- Peripheral information is updated in the system after calling back njUserMain. Please note that peripheral information isn't updated in the case that it doesn't return from njUserMain.
- Joystick must be set up properly at the control panel.
- Only two joysticks can be used at present version.

## RELATED TOPICS

[njPrintPeripheralInfo](#)

[NjWinDef.h](#)

## njPrintPeripheralInfo

Displays peripheral condition.

---

### FORMAT

```
#include <NinjaWin.h>
void njPrintPeripheralInfo(Int loc, Uint32 port)
Int loc
Uint32 port
```

### PARAMETERS

**loc**

Displayed text coordinate

**port**

Peripheral port number

NJD_PORT_SYSKEYBOARD	System keyboard
NJD_PORT_SYSMOUSE	System mouse
NJD_PORT_JOYSTICK1	Joystick port 1
:	
NJD_PORT_JOYSTICK16	Joystick port 16

### RETURN VALUE

None

#### FUNCTION

- Displays main members of peripheral structure in text format.

## EXAMPLE

```
Sint32 njUserMain(void)
{
    njPrintPeripheralInfo(NJM_LOCATION( 0, 0), NJD_PORT_SYSKEYBOARD);
    njPrintPeripheralInfo(NJM_LOCATION(20, 0), NJD_PORT_SYSMOUSE);
    njPrintPeripheralInfo(NJM_LOCATION(40, 0), NJD_PORT_JOYSTICK1);
    njPrintPeripheralInfo(NJM_LOCATION(60, 0), NJD_PORT_JOYSTICK2);
    return NJS_USER_CONTINUE;
}
```

## NOTES

## RELATED TOPICS

[njGetPeripheral](#)

[NjWinDef.h](#)

