

SEGA™

Ninja Guide



Katana Ninja Guide

Table of Contents

01. View Function	NGD-1
Initialization Method	NGD-1
1. Use njInitView().	NGD-1
Set the view as follows:	NGD-1
2. Directly set VIEW structure members.	NGD-1
1. When performing relative operations:	NGD-1
2. When performing absolute operations:	NGD-2
View Movement and Rotation	NGD-2
Example:	NGD-2
Structures	NGD-2
02. Texture Guide	NGD-3
Modification History	NGD-3
Ver.0.04	NGD-3
Ver.0.05	NGD-3
ver0.06	NGD-3
Overview	NGD-4
Terminology	NGD-4
Textures	NGD-4
Texture List	NGD-4
Texture Number	NGD-4
Global Index Number	NGD-4
Current Texture List	NGD-4
Current Texture	NGD-4
PVR Format	NGD-4
U, V Coordinates	NGD-4
Aspect Ratio	NGD-4
Mipmap	NGD-4
LOD (level of detail)	NGD-5

Texture Memory	NGD-5
Cache	NGD-5
Texture Information Area	NGD-5
Cache Information Area	NGD-5
Category Code	NGD-5
Stride Value	NGD-5
Creating Textures	NGD-6
Overview	NGD-6
PVR Format	NGD-6
Category Code	NGD-7
Twiddled, Twiddled Mipmap Format	NGD-7
VQ,VQ Mipmap Format (Vector Quantization)	NGD-7
Palettize4, Palettize4 Mipmap Format, Palettize8, Palettize8 Mipmap Format	NGD-7
Rectangle Format	NGD-7
Stride Format	NGD-8
Color Format	NGD-8
Normal Texture Color Format	NGD-8
YUV422 format	NGD-8
Bump format	NGD-8
ARGB8888 format	NGD-8
Texture formats supported by NINJA	NGD-8
Memory	NGD-9
Overview	NGD-9
Texture Memory	NGD-9
Cache	NGD-9
Loading Textures	NGD-10
Overview	NGD-10
Flowchart of Texture Loading	NGD-10
Creating a Texture List	NGD-11
1. Define a texture name structure with as many elements as there are textures.	NGD-11
2. Use the texture name structures created in part 1 to set up a texture list structure	NGD-12
Texture Numbers	NGD-13
Global Index Number	NGD-14
Texture Load Error	NGD-14
NJD_TEXERR_FILEOPEN	NGD-14
NJD_TEXERR_EXTND	NGD-14
NJD_TEXERR_HEADER	NGD-14
NJD_TEXERR_FILELOAD	NGD-15
NJD_TEXERR_SURFACE	NGD-15
NJD_TEXERR_MAINMEMORY	NGD-15
NJD_TEXERR_TEXMEMLOAD	NGD-15
Global Index Error	NGD-15
Frame Buffer Texture	NGD-15
Texture Functions, Structures, and Definitions	NGD-16
Overview	NGD-16

njInitTexture	NGD-16
Format	NGD-16
Parameter	NGD-16
*addr	NGD-16
n	NGD-16
Return Value	NGD-16
Function	NGD-16
Note	NGD-16
njLoadTexture	NGD-17
Format	NGD-17
Parameter	NGD-17
*texlist	NGD-17
Return Value	NGD-17
Success	NGD-17
Failure	NGD-17
Function	NGD-17
Note	NGD-17
njLoadTextureNum	NGD-18
Format	NGD-18
Parameter	NGD-18
n	NGD-18
Return Values	NGD-18
Success	NGD-18
Failure	NGD-18
Function	NGD-18
Note	NGD-18
njSetTexture	NGD-19
Format	NGD-19
Parameter	NGD-19
*texlist	NGD-19
Return Value	NGD-19
Success	NGD-19
Failure	NGD-19
Function	NGD-19
Notes	NGD-19
njSetTextureNum	NGD-20
Format	NGD-20
Parameter	NGD-20
n	NGD-20
Return Value	NGD-20
Success	NGD-20
Failure	NGD-20
Function	NGD-20
Notes	NGD-20

njSetTextureNumG	NGD-21
Format	NGD-21
Parameter	NGD-21
Return Value	NGD-21
Success	NGD-21
Failure	NGD-21
Function	NGD-21
Notes	NGD-21
njLoadCacheTextureNum	NGD-22
Format	NGD-22
Parameter	NGD-22
n	NGD-22
Return Value	NGD-22
Success	NGD-22
Failure	NGD-22
Function	NGD-22
Notes	NGD-22
njLoadCacheTextureNumG	NGD-23
Format	NGD-23
Parameter	NGD-23
globalIndex	NGD-23
Return Value	NGD-23
Success	NGD-23
Failure	NGD-23
Function	NGD-23
Notes	NGD-23
njReleaseTextureAll	NGD-24
Format	NGD-24
Parameter	NGD-24
Return Value	NGD-24
Function	NGD-24
Notes	NGD-24
njReleaseTexture	NGD-25
Format	NGD-25
Parameter	NGD-25
*texlist	NGD-25
Return Value	NGD-25
Success	NGD-25
Failure	NGD-25
Function	NGD-25
Notes	NGD-25
njReleaseTextureNum	NGD-26
Format	NGD-26
Parameter	NGD-26
n	NGD-26

Return Value	NGD-26
Success	NGD-26
Failure	NGD-26
Function	NGD-26
Notes	NGD-26
njReleaseTextureNumG	NGD-27
Format	NGD-27
Parameter	NGD-27
Return Value	NGD-27
Success	NGD-27
-1	NGD-27
Function	NGD-27
Notes	NGD-27
njReleaseCacheTextureAll	NGD-28
Format	NGD-28
Parameter	NGD-28
Return Value	NGD-28
Function	NGD-28
Notes	NGD-28
njReleaseCacheTextureNum	NGD-29
Format	NGD-29
Parameter	NGD-29
n	NGD-29
Return Value	NGD-29
Success	NGD-29
Function	NGD-29
Notes	NGD-29
njReleaseCacheTextureNumG	NGD-30
Format	NGD-30
Parameter	NGD-30
globalIndex	NGD-30
Return Value	NGD-30
Success	NGD-30
Failure	NGD-30
Function	NGD-30
Notes	NGD-30
njGetTextureNumG	NGD-31
Format	NGD-31
Parameter	NGD-31
Return Value	NGD-31
Success	NGD-31
Failure	NGD-31
Function	NGD-31
Notes	NGD-31

njCalcTexture	NGD-32
Format	NGD-32
Parameter	NGD-32
Return Value	NGD-32
Function	NGD-32
Notes	NGD-32
njExitTexture	NGD-33
Format	NGD-33
Parameter	NGD-33
Return Value	NGD-33
Function	NGD-33
Notes	NGD-33
njSetTexturePath	NGD-34
Format	NGD-34
Parameter	NGD-34
Return Value	NGD-34
Function	NGD-34
Notes	NGD-34
njSetTextureInfo	NGD-35
Format	NGD-35
Parameter	NGD-35
Return Value	NGD-35
Functions	NGD-35
Note	NGD-35
Color format	NGD-35
Category code	NGD-35
njSetTextureName	NGD-36
Format	NGD-36
Parameter	NGD-36
Return value	NGD-36
Functions	NGD-36
Notes	NGD-36
njReloadTextureNum (New Function)	NGD-37
Overview	NGD-37
Syntax	NGD-37
Parameters	NGD-37
Return Value	NGD-37
Success	NGD-37
Failure	NGD-37
Purpose	NGD-37
Remarks	NGD-37
njReloadTextureNumG (New Function)	NGD-38
Overview	NGD-38
Syntax	NGD-38
Parameters	NGD-38

Return Value	NGD-38
Success	NGD-38
Failure	NGD-38
Purpose	NGD-38
Remarks	NGD-38
njSetRenderWidth (New Function)	NGD-39
Overview	NGD-39
Syntax	NGD-39
Parameter	NGD-39
Return Value	NGD-39
Purpose	NGD-39
Remarks	NGD-39
NjFrameBufferBmp (New Function)	NGD-39
Overview	NGD-39
Syntax	NGD-39
Parameter	NGD-39
Unit8*filename	NGD-39
Return Value	NGD-40
Purpose	NGD-40
Remarks	NGD-40
Deleted functions	NGD-40
Texture Structures	NGD-40
Texture Definitions	NGD-41
Used with nWidth, nHeight	NGD-41
Used with attr	NGD-41
Used with Type, color format	NGD-42
Category code	NGD-42
Texture error code (new addition)	NGD-42
Acquire texture memory size (used with njCalcTexture)	NGD-42
Sample Program	NGD-43
Overview	NGD-43
Sample	NGD-43
Ex. 1 : Display of PVR texture file	NGD-43
Ex. 2: Load a texture from memory and display it.	NGD-44
Ex. 3: Load file from cache and display texture	NGD-45
03. NINJA LIGHT	NGD-47
How to set LIGHT	NGD-47
>> void njCreateLight(NJS_LIGHT*, Int)	NGD-47
>> void njDeleteLight(NJS_LIGHT*)	NGD-47
>> void njLightOff(NJS_LIGHT*)	NGD-47
>> void njLightOn(NJS_LIGHT*)	NGD-48
>> void njMultiLightMatrix(NJS_LIGHT*, NJS_MATRIX*)	NGD-48
>> void njSetLight(NJS_LIGHT*)	NGD-48
>> void njSetLightAlpha(NJS_LIGHT*, Float)	NGD-48

>> void njSetLightAngle(NJS_LIGHT*, NJS_Angle, NJS_Angle)	NGD-48
>> void njSetLightColor(NJS_LIGHT*, Float, Float, Float)	NGD-49
>> void njSetLightDirection(NJS_LIGHT*, Float, Float, Float)	NGD-49
>> void njSetLightIntensity(NJS_LIGHT*, Float, Float, Float)	NGD-49
>> void njSetLightLocation(NJS_LIGHT*, Float, Float, Float)	NGD-49
>> void njSetLightRange(NJS_LIGHT*, Float, Float)	NGD-50
>> void njSetUserLight(NJS_LIGHT*, NJF_LIGHT_FUNC*)	NGD-50
>> void njUnitLightMatrix(NJS_LIGHT*)	NGD-50
>> void njTranslateLightV(NJS_LIGHT*, NJS_VECTOR*)	NGD-50
>> void njTranslateLight(NJS_LIGHT*, Float, Float, Float)	NGD-50
>> void njRotateLightX(NJS_LIGHT*, NJS_Angle)	NGD-51
>> void njRotateLightXYZ(NJS_LIGHT*, NJS_Angle, NJS_Angle,	NGD-51
>> void njRotateLightY(NJS_LIGHT*, NJS_Angle)	NGD-51
>> void njRotateLightZ(NJS_LIGHT*, NJS_Angle)	NGD-51
Macro	NGD-52
How to Use	NGD-52
Example1	NGD-52
Example2	NGD-53
Example3	NGD-54
LIGHTstructure Specification	NGD-54
The members of NJS_LIGHT structure	NGD-55
The members of NJS_LIGHT_ATTR structure	NGD-55
(Example:Spot light)	NGD-57
(Example:point light source)	NGD-57
The members of NJS_LIGHT_CAL structure	NGD-58

04. Ninja Model and Motion NGD-59

Model Structures	NGD-59
Diagram of Structure	NGD-60
Description of Structures	NGD-61
Float, Angle	NGD-61
Object structure	NGD-61
Explanation of evalflags	NGD-62
Point structure	NGD-62
Texture name structure	NGD-63
Texture list structure	NGD-64
Material structure	NGD-64
Meshset structure	NGD-64
Model structures	NGD-65
Construction of a Model	NGD-66
Diagram of Structure	NGD-66
Meshsets	NGD-67
For a triangular polygon list (NJD_MESHSET_3)	NGD-68
For a quadrilateral polygon list (NJD_MESHSET_4)	NGD-68

For a contiguous polygon list (NJD_MESHSET_TRIMESH)	NGD-69
For an N-sided polygon list (NJD_MESHSET_N)	NGD-70
Construction of a Texture	NGD-71
Diagram of Structure	NGD-71
Overview of Texture Processing	NGD-72
Memory-type Textures and the Texture Cache	NGD-73
Explanation of the structure of textures	NGD-73
Ninja Attributes	NGD-74
Texture Format	NGD-79
Motion Structures	NGD-81
Diagram of Structure	NGD-81
Explanation of Structures	NGD-82
Action structure	NGD-82
Motion structure	NGD-82
NJS_MDATA1 to 4 structures	NGD-83
Key structures	NGD-83
Construction of a Motion	NGD-84
Diagram of Structure	NGD-84
Explanation of Structure	NGD-85
05. Scroll Guide	NGD-87
Revision Information	NGD-87
Ver.0.04	NGD-87
Ver.0.05	NGD-87
Image Units as Related to Scrolling	NGD-87
Overview	NGD-87
Image Units	NGD-87
Pixel	NGD-87
Cell	NGD-87
Map	NGD-88
Scroll Rotation, Resizing, and Movement	NGD-88
Overview	NGD-88
Scroll Rotation, Resizing, and Movement	NGD-88
Scroll Programming	NGD-90
Overview	NGD-90
Example of Programming a Scroll	NGD-90
Color	NGD-92
Overview	NGD-92
Color Mode	NGD-92
NJD_COLOR_MODE_FLAT_TEXTURE	NGD-92
NJD_COLOR_MODE_FLAT_TEXTURE_TRANS	NGD-92

Scroll function, Structures, and Definitions	NGD-93
Overview	NGD-93
Scroll-related Functions	NGD-93
njDrawScroll	NGD-93
Format	NGD-93
Parameters	NGD-93
Return value	NGD-93
Function	NGD-93
Notes	NGD-93
Scroll-related Structure	NGD-94
Scroll-related Definitions	NGD-94
Texture Structures for Use in Cell Programming	NGD-95
06. Nindows Tutorial	NGD-97
1. Nindows Tutorial	NGD-97
1.1 Summary	NGD-97
1.2 Special Features of Nindows	NGD-97
2 Creating a Simple Nindows Application	NGD-98
2.1 Integrating Nindows	NGD-98
Preparing the Ninja application.	NGD-98
Include the Nindows header file	NGD-98
Call the Nindows initialization function	NGD-98
Call the function to execute Nindows	NGD-98
Call the function to exit Nindows	NGD-98
Linking the Nindows Library	NGD-98
2.2 Description of Functions used in Integrating Nindows	NGD-99
3. Using Nindows and Nindows Utilities	NGD-102
3.1 Using Nindows	NGD-102
System Menu	NGD-102
3.2 Nindows Utilities	NGD-103
Ninja Info Window	NGD-103
Texture Viewer Window	NGD-104
Peripheral Info Window	NGD-105
Window Info Window	NGD-105
Debugging Window	NGD-106
Performance Meter Window	NGD-107
Changing Fonts	NGD-107
4. Windows	NGD-108
4.1 Summary	NGD-108
4.1.1 Types of Windows and Window Classes	NGD-108
4.2 Creating a Window	NGD-108
4.3 Creating a Child Window	NGD-109
4.4 Window Related Parameters	NGD-110

4.5 Description of Window Support Functions	NGD-110
Nindows API	NGD-111
Callback Functions	NGD-112
4.4 Samples and a Description of Window Support Functions	NGD-113
Nindows API	NGD-115
Structure	NGD-122
5. Scroll Windows	NGD-124
5.1 Summary	NGD-124
5.2 Creating a Scroll Window	NGD-124
5.3 Description of Functions Used to Create a Scroll Window	NGD-125
Nindows API	NGD-125
6. Edit Windows	NGD-128
6.1 Summary	NGD-128
6.2 Creating and Using an Edit Window	NGD-128
6.3 Description of Functions Used in Creating Edit Windows	NGD-129
Nindows API	NGD-130
6.4 Description of Functions Used in Nindows' Debug Window Utility	NGD-131
7. Scrollbar Controls	NGD-132
7.1 Summary	NGD-132
7.2 Creating Scrollbar Controls	NGD-132
7.3 Description of Functions Used in Creating Scrollbar Controls	NGD-133
Nindows API	NGD-134
Structure	NGD-134
7.4 Creating Scrollbar Controls that Use Low-level Scrollbar Functions	NGD-136
7.5 Description of Low-level Scrollbar Functions	NGD-137
Nindows API	NGD-137
8. Button Controls	NGD-141
8.1 Summary	NGD-141
8.2 Creating a Button Control	NGD-141
8.3 Button Validity and Invalidity	NGD-142
8.4 Description of Functions for Button Controls	NGD-143
Nindows API	NGD-144
Callback Function	NGD-145
9. Menus	NGD-146
9.1 Summary	NGD-146
9.2 Creating and Entering Menu Tables	NGD-146
9.3 Menu Callback Functions	NGD-149
9.4 Checkmarks	NGD-149
Display checkmark	NGD-149
Hide checkmark	NGD-149
9.5 Description of Functions for Entering User Menus	NGD-151
Nindows API	NGD-151
Callback Function	NGD-152
Structure	NGD-152

9.6 Creating Popup Menus	NGD-153
9.6.1 Creating a Simple Popup Menu	NGD-153
9.6.2 Creating a Popup Menu that Stays on the Screen	NGD-154
9.7 Description of Functions Used in Creating Popup Menus	NGD-154
10. Mouse	NGD-155
10.1 Summary	NGD-155
10.2 Getting Mouse Information	NGD-155
10.3 Description of Functions Used for Getting Mouse Information	NGD-156
Nindows API	NGD-157
Structure	NGD-158
11. Fonts	NGD-159
11.1 Overview	NGD-159
11.2. Description of Font Functions	NGD-159
Nindows API	NGD-159
11.3. Problems with Changing Fonts	NGD-159



01. View Function

Initialization Method

View initialization must be completed before the `njSetView()` function is executed.

There are two methods of initialization.

1. Use `njInitView()`.

Set the view as follows:

Current position of viewpoint: $(px, py, pz) = (0,0,0)$

Current orientation of viewpoint: $(vx, vy, vz) = (0,0,-1)$

Current tilt of viewpoint: (roll, tilt versus Z axis of viewline) 0 degrees

Base position of viewpoint: $(apx, apy, apz) = (0,0,0)$

Base orientation of viewpoint: $(avx, avy, avz) = (0,0,-1)$

Base tilt of viewpoint: (aroll, tilt versus Z axis of viewline) 0 degrees

View matrix = Unit matrix

2. Directly set VIEW structure members.

The following settings must be made:

1. When performing relative operations:

`px, py, pz`

`vx, vy, vz`

`roll`

2. When performing absolute operations:

```
px, py, pz
vx, vy, vz
roll
```

- After setting direct values for the above, execute `void njSetBaseView(NJS_VIEW *v)`.
- Or, set the same respective values in the following:

```
apx, apy, apz
avx, avy, avz
aroll
```

- The view matrix settings are not needed.

 **Caution:** The viewline vector must be converted to unit vectors.

View Movement and Rotation

- All `nj*Relative()` and `nj*Absolute()` functions must be executed before the `njSetView()` function.

Example:

```
Initialize view
:
:
while(1){
    njSetView();
    :
    :
    :
    Draw
    :
    : <----- Execute nj*Relative() and nj*Absolute(), and then proceed to the
next viewpoint.
    :
}
```

 **Important:** In the flow of the program, be certain to execute any `nj*Relative()` or `nj*Absolute()` function before the `njSetView()` function.

Structures

```
typedef struct {
    Float   px,py,pz;           // Current position of viewpoint
    Float   vx,vy,vz;         // Current orientation of viewpoint (vector)
    Angle   roll;             // Current tilt versus Z axis of viewline
    Float   apx,apy,apz;      // Base position of viewpoint
    Float   avx,avy,avz;     // Base orientation of viewpoint (vector)
    Angle   aroll;           // Base tilt versus Z axis of viewline
    NJS_MATRIX m;           // View matrix
} NJS_VIEW;
```



02. Texture Guide

Modification History

Ver.0.04

- njInitCacheTexture function was deleted.
- njLoadCacheTexture function was added.
- njCalcTexture function was modified.
- The way to use Texture Cache was modified.
- PVR texture format was added.

Ver.0.05

- 2.2: The figure of PVR format was changed.
- 4.2: Fixed the bug of "Flowchart of Texture Loading".
- 4.6 "Texture Load Error" was added.
- 5.4: "Texture Error Definitions" was added.
- 6.2: Fixed the bug in Example3 (Cache sample).

ver0.06

- 1.1: Added description of the Stride value.
- 2.3.4: Changed the description of the RECTANGLE format.
- 2.3.5: Changed the description of the STRIDE format.
- 2.4: Changed diagram.
- 4.2: Modified the Texture Load Flow diagram.
- 4.3: Added NJD_TEXATTR_TYPE_FRAMEBUFFER to the load destinations for texture list generation.
- 4.6: Added NJD_TEXERR_GLOBALINDEX to the texture errors.
- 4.7: Added frame buffer texture.
- 5.2: Changed the texture-related njCalcTexture function and added 6 more functions.
- 5.4: Changed and added related texture definitions.

Overview

This chapter explains the meanings of terms applying to making textures with Ninja.

Terminology

Textures

In Ninja, the term “texture” refers to all images applied to 2D graphics, 3D graphics, sprites, scrolls, models, etc. Ninja can use textures of the following lengths and widths: 1024, 512, 256, 128, 64, 32, 16, 8.

Texture List

A list of all the textures used at a given time is called a texture list. The basic concept in Ninja is to manipulate textures at the texture list level. Texture list creation is covered in Chapter 4

Texture Number

Number assigned in ascending order to textures in a texture list, 0, 1, 2...etc. Details will be covered at a later date.

Global Index Number

Number applied consistently to a given texture throughout source code. Textures with the same global index number are considered to be the same texture.

Current Texture List

Designation for the texture list being operated on by a texture function.

Current Texture

Designation for the texture in the current texture list being operated by a texture function. Many of the texture functions perform texture manipulations on the current texture.

PVR Format

Format for texture files that can be loaded with Ninja.

U, V Coordinates

Coordinates within a texture are designated U (horizontal) and V (vertical). Both U and V range from 0 to 1, even if the aspect ratio between U and V varies.

Aspect Ratio

The ratio of horizontal to vertical in a texture is called the aspect ratio.

Mipmap

Designation for a set of textures which are represented by the same texture map order.

LOD (level of detail)

The mipmap level.

Texture Memory

Memory used for texture storage.

Cache

As many textures (more than the portion for which the texture memory can load) are used, textures are preloaded into a portion of main memory. This is referred to as the cache, and can be used most effectively when holding textures that are used and replaced frequently.

Texture Information Area

The area within Ninja where information about textures loaded into texture memory is stored.

Cache Information Area

The area within Ninja where information about textures loaded into the cache is stored.

Category Code

The texture format which can be used in Ninja. The following texture formats can be used in Ninja: Twiddled, Twiddled Mipmap, VQ, VQ Mipmap, Palettize4, Palettize4 Mipmap, Palettize8, Palettize8 Mipmap, Rectangle, Stride. Refer to the Creating Textures section below for details.

Stride Value

Specify when the STRIDE format texture is used by NINJA. Acceptable values are multiples of 32 between 32 and 992.

Creating Textures

Overview

This chapter describes the category code and color format which can be used in Ninja.

PVR Format

Global Index Tag	ID Area <i>GBIX</i>	4byte
Global Index Tag	Byte Number to the Next Tag	4byte
Global Index Tag	Global Index	4byte

PVR Format Tag	ID Area <i>PVRT</i>	4byte
PVR Format Tag	Byte Number to the Next Tag	4byte
PVR Format Tag	Texture Attribute	4byte
PVR Format Tag	Width	2byte
PVR Format Tag	Length	2byte
Each Data		

There are two PVR formats: both with and without global index header. The category code and color format are specified as the texture attribute.

Category Code

The texture formats which can be used in Ninja are called *category code*. The details for each category code is as follows.

Twiddled, Twiddled Mipmap Format

Twiddled format is the basic format of Ninja. In this format, the inside of the texture is optimized and reallocated in order to load each filter and texture. For this reason, the inside of the texture is not lined in the raster order. Also, textures must be square for Twiddled format.

0	2	8	10	32				128	130						
1	3	9	11					129	131						
4	6	12	14												
5	7	13	15												
16	18														
17	19														
20															
			31				63								
64								192							

VQ,VQ Mipmap Format (Vector Quantization)

VQ texture is the compression texture format of high compression rate. VQ textures create the image using the color table which is called Codebook and Index which shows the location of the codebook. VQ textures are not supported yet by Ninja.

Palettize4, Palettize4 Mipmap Format, Palettize8, Palettize8 Mipmap Format

There are two types of Palettized textures: 4bpp mode and 8bpp mode. These two can be used simultaneously. This format is the same as Twiddled format on the memory. Not supported yet by Ninja.

Rectangle Format

Different sizes can be specified for the width and length of Rectangle texture. Mipmap can not be used for Rectangle textures. Also, the performance of Rectangle format is lower than the one of the Twiddled texture.

Stride Format

As a special form of RECTANGLE rendering is possible in this area, and it can be used as a texture. When using a STRIDE format texture, a STRIDE value must be specified. NINJA uses the `njSetRenderWidth` function. The STRIDE format texture determines the texel using the following addressing method.

$$\text{Addr} = U + V * \text{Stride}$$

For example, when a 640 × 480 area is to be used as a Stride texture in a 1024 × 1024 texture area, specify 640 as the Stride value. In this case, the UV value is (U,V) = (0,0) ñ (0.625f,0.46875f) to apply to the full size screen.

Color Format

The color formats which can be used in Ninja are described as follows.

Normal Texture Color Format

The color formats which can be used in Ninja normally are ARGB1555, ARGB4444, RGB565.

YUV422 format

1 pixel can be displayed by 8bit in this format. Not supported yet by Ninja.

Bump format

Texture format for bump mapping. Not supported yet by Ninja.

ARGB8888 format

The format for Palettizing. Not supported yet by Ninja.

Texture formats supported by NINJA

	ARGB1555	RGB565	ARGB444	YUV422	Bump	ARGB8888
Twiddled	A	A	A	F	F	X
Twiddled MM	A	A	A	F	F	X
VQ	F	F	F	F	F	X
VQ MM	F	F	F	F	F	X
Palettized 4,8	F	F	F	X	X	F
Palettized MM	F	F	F	X	X	F
Rectangle	A	A	A	F	F	X
Stride	A	A	A	F	F	X

A: Available **F:** Available in future version **X:** Not available

Memory

Overview

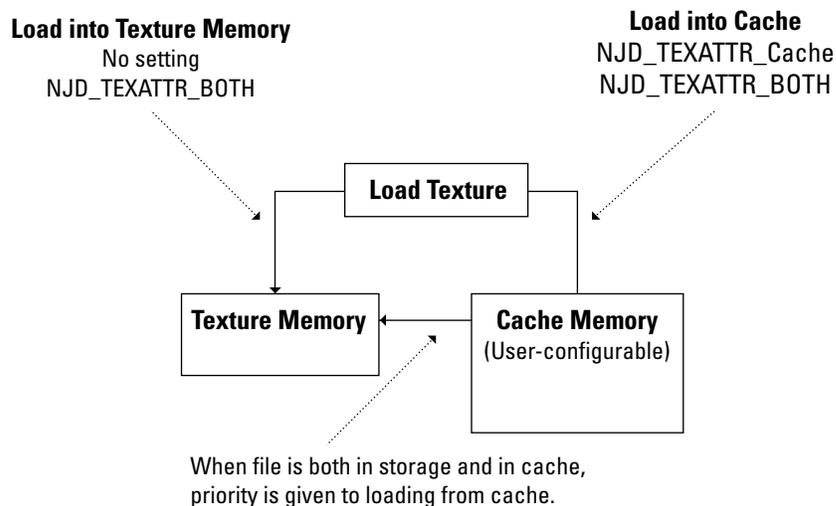
Ninja uses both texture memory and cache memory for loading textures. This section explains the two types of memory.

Texture Memory

The texture memory is the area reserved for textures. The texture memory area can be read.

Cache

In order to make effective use of the texture memory, users to configure an area of main memory for texture loading, known as the cache. Ninja gives priority to loading textures stored in the cache. To load textures into the cache, set the texture's attribute to cache at time of loading. Note that textures already loaded into main memory are not loaded into the cache; only textures in file storage are loaded into the cache.

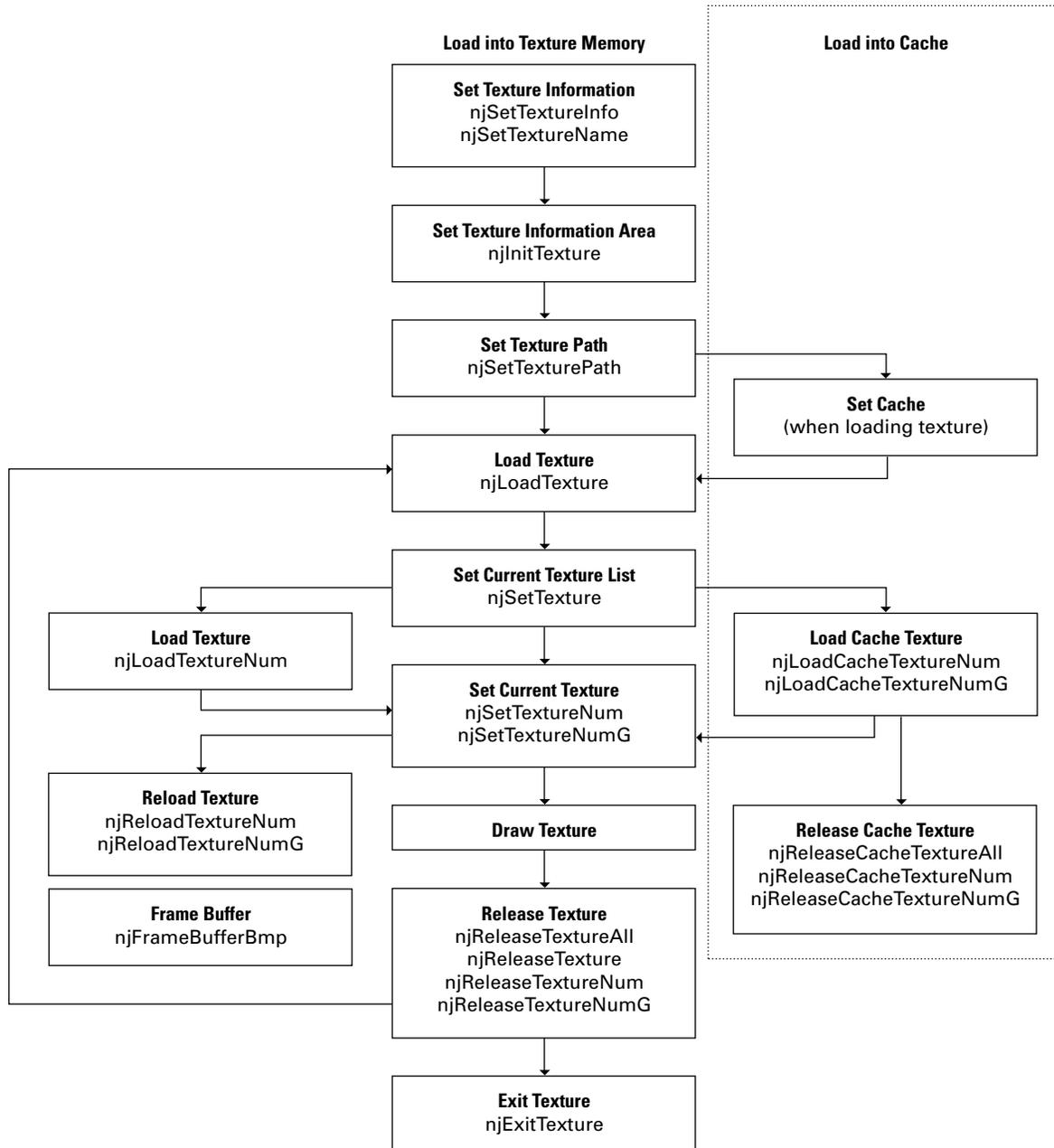


Loading Textures

Overview

Now we will try using texture functions to load a texture. We will begin with a general flowchart of the texture loading process, followed by explanations of how to create texture lists, texture numbers, and global index numbers.

Flowchart of Texture Loading



Note: When executing `njLoadTextureNum` and run `njSetTexture`, set the current texture list.

Creating a Texture List

In Ninja, the texture list is the fundamental part of the texture manipulation. This section describes texture list settings

1. Define a texture name structure with as many elements as there are textures.

NJS_TEXNAME structure

```
void          *filename
Uint32        attr
Uint32        texaddr
```

**filename*

NJS_TEXINFO pointer, used when loading textures from designated memory; sets file name for PVR format texture files to string

attr

It sets source and destination of texture load. It takes the OR of the various tags

>Load Source

NJD_TEXATTR_TYPE_FILE

Load PVR format file. Designate file name with **filename*.

NJD_TEXATTR_TYPE_MEMORY

Load from memory. Designate NJS_TEXINFO pointer with **filename*

* Load Destination (will load into texture memory if not specified)

NJD_TEXATTR_TYPE_FRAMEBUFFER (NEW)

Use a frame buffer as a texture

>Load Destination (will load into texture memory if not specified)

NJD_TEXATTR_CACH

Load only into cache memory

NJD_TEXATTR_BOTH

Load into both texture memory and cache memory

texaddr

It sets the global index for the memory texture. It becomes the pointer for the internal table after texture loading.

NJS_TEXINFO

```
void*                texaddr;
NJS_TEXSURFACE      texsurface;
```

texaddr

It is used to reserve the texture in the texture memory.

texsurface

It is the format to pass the data to the inside.

NJS_TEXSURFACE structure

```
Uint32      Type;
Uint32      BitDepth;
Uint32      PixelFormat;
Uint32      nWidth;
Uint32      nHeight;
Uint32      TextureSize;
Uint32      fSurfaceFlags;
Uint32      *pSurface;
```

Type

The color format and category code are set for the memory texture.

nWidth

The width of the texture is set for the memory texture.

nHeight

The length of the texture is set for the memory texture.

As for other members, these are set in the load function.

When the load source is memory, the NJS_TEXINFO structure needs to be set:

2. Use the texture name structures created in part 1 to set up a texture list structure

NJS_TEXLIST

```
NJS_TEXNAME      *textures;
Uint32           nbTexture;
```

textures

Sets pointer to NJS_TEXNAME structure, which holds texture information.

nbTexture: number of textures

nbTexture

Number of textures

Ex.: file01.pvr and the memory texture image are specified as the texture is as follows.

```
extern Uint16 Image[];

NJS_TEXINFO Info;
NJS_TEXNAME texname[2];
NJS_TEXLIST texlist={texname,2};

/* Memory texture      Image
   Category code      TWIDDLED
   Color format       ARGB1555
   Size               256x256
*/
njSetTextureInfo(&Info, Image, NJD_TEXFMT_TWIDDLED | NJD_TEXFMT_ARGB_1555, 256, 256);

/* Set the file \file0.pvr\ for texname[0] */
njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE);

/* Set the memory texture image for texname[1] */
njSetTextureName(&texname[1], &Info, 1, NJD_TEXATTR_TYPE_MEMORY);

/* The initial of the texture */
njInitTexture(texmemlist, 2);

/* Load texture */
njLoadTexture(&texlist);
```

Texture Numbers

For the current texture list, assign texture numbers 0, 1, 2 ... etc. to the structure NJS_TEXNAME created in 6.3, in setting order: 0

```
NJS_TEXNAME texname[]={      {"file0.pvr",,,,},      /* texture number0 */
                           {"file1.pvr",,,,},      /* texture number1 */
                           {"file2.pvr",,,,},      /* texture number2 */
                           {"file3.pvr",,,,},      /* texture number3 */
                           :
                           {"filen.pvr",,,,};      /* *texture numbern */
```

The texture numbers used in Ninja texture functions are taken from the texture numbers in the current texture list.

Global Index Number

Ninja assigns numbers which apply globally throughout an application to ensure that a given texture only gets loaded once into texture memory, even when working with multiple texture lists. These numbers are called global index numbers. Textures with the same global index number are treated as the same texture. Global index numbers apply to all textures, including those for 2D and 3D graphics, sprites, scrolls, and models, so be careful that one number gets assigned to only one texture. Conversely, if you apply different global index numbers to the same texture, the textures which match up will be loaded into texture memory.

In the file of PVR format, there is a chunk inside to hold the global index. The global index of the PVR format textures are managed by the tools.

Assign global index numbers from 0 to 0xFFFFFFFF. As the numbers from 0xFFFFFFFF0 to 0xFFFFFFFFF is used by the system, do not use it as a global index number assignment.

Texture Load Error

The data after load request is stored in the texture memory list (NJS_TEXMEMLIST) which is set in njInitTexture by the user. The following data is stored in the texture memory list.

UInt32	globalIndex;	Global Index
UInt32	texaddr;	BIT_0: Load into texture memory BIT_1: Load into cache
NJS_TEXINFO	texinfo;	Texture info structure
UInt16	count;	Use number of times
UInt16	dummy;	Error code (New addition)

In case an error is found when textures are loaded, the following error codes are set to the dummy.

```
#define NJD_TEXERR_OTHER          (1)    //Other errors
#define NJD_TEXERR_FILEOPEN      (2)    //File open error
#define NJD_TEXERR_EXTND         (3)    //Extension error
#define NJD_TEXERR_HEADER       (4)    //Header error
#define NJD_TEXERR_FILELOAD     (5)    //File load error
#define NJD_TEXERR_SURFACE      (6)    //Surface creation error
#define NJD_TEXERR_MAINMEMORY   (7)    //Main memory malloc error
#define NJD_TEXERR_TEXMEMLOAD   (8)    //Texture memory load error
#define NJD_TEXERR_GLOBALINDEX  (9)    //Globalindex error
```

NJD_TEXERR_FILEOPEN

This error appears when files cannot be opened as they are not in the specified location.

NJD_TEXERR_EXTND

This error appears when the extension of the file is not `.pvr`.

NJD_TEXERR_HEADER

The header of the texture file is not correct. This error appears when the way to use GBIX tag and PVRT tag is not correct.

NJD_TEXERR_FILELOAD

This error appears when files cannot be loaded or the data is smaller than the size expected.

NJD_TEXERR_SURFACE

This error appears when the area to load textures can not be reserved in the texture memory. Also, this error appears when the texture size is too big or the texture which can not be loaded is specified.

NJD_TEXERR_MAINMEMORY

This error appears when the area for the work buffer can not be reserved in the texture load function.

NJD_TEXERR_TEXMEMLOAD

This error appears when textures can not be loaded into the texture memory.

This error does not appear usually (as NJ_TEXERRR_SURFACE is supposed to appear before this error).

Global Index Error

This error is output when an invalid global index is specified, or when a global index could not be obtained.

Frame Buffer Texture

A frame buffer can be specified as a texture load destination when loading a texture, so the frame buffer is used for the texture.

Of the two frame buffers, the displayed frame buffer can be used as the texture for the next drawing. This means that the texture applied as a frame buffer texture is the previous frame buffer.

To use a frame buffer in practice, make NJS.TEXTATTR.FRAMEBUFFER the source texture in attr within the NJS.TEXTNAME structure. This disables the filename. Also, when using a frame buffer texture, the Stride value (set by the njSetRenderWidth function) must be set to the width of the screen.

Although the texture size when using a frame buffer texture is 1024 × 1024, no memory other than the frame buffer is actually used for the texture. Also, since a frame buffer texture uses only the previous frame buffer data, performance is not degraded by using a frame buffer texture.

Frame buffer modes that can be used as frame buffer textures are NJD_FRAMEBUFFER_MODE_RGB565, NJD_FRAMEBUFFER_MODE_RGB555 and NJD_FRAMEBUFFER_MODE_ARGB1555.

When using a frame buffer texture, UV is 0 to 1.0:

$(U,V) = (0,0) \sim (\text{Width}/1024, \text{Height}/1024)$



Note: Width and Height are the horizontal and vertical screen size.

Texture Functions, Structures, and Definitions

Overview

This chapter covers Ninja texture functions, texture structures, and texture definitions

njInitTexture

Set texture information area

Format

```
#include <Ninja.h>
void njInitTexture(*addr,n);
NJS_TEXMEMLIST *addr
Uint32 n
```

Parameter

*addr

NJS_TEXMEMLIST structure pointer to area of n elements

n

number of textures

Return Value

none

Function

By setting an NJS_TEXMEMLIST structure area of a size n, where n is the number of textures to be used, to a pointer to addr, this function makes it into an area for holding texture information. Be sure to execute this function before loading textures

Note

The memory area defined in this function is used internally by texture-related functions.

njLoadTexture

Load texture

Format

```
#include <Ninja.h>
 Sint32 njLoadTexture(texlist);
 NJS_TEXLIST *texlist
```

Parameter

***texlist**

NJS_TEXLIST structure pointer

Return Value

Success

1

Failure

-1

Function

The texture file specified in the texlist structure is loaded as texture memory, cache memory or the frame buffer texture.

Note

Before executing this function, it is necessary to run njInitTexture first.

njLoadTextureNum

Load textures by texture number.

Format

```
#include <Ninja.h>
Sint32 njLoadTextureNum(n);
Uint32 n
```

Parameter

n

texture number of current texture list

Return Values

Success

1

Failure

-1

Function

Load the texture in the current texture list with texture number *n* into texture memory or cache memory. If texture number *n* is not in current texture list, function returns an error

Note

Before running this function, it is necessary to run `njInitTexture` and `njSetTexture`.

njSetTexture

Set current texture list

Format

```
#include <Ninja.h>
 Sint32 njSetTexture(texlist);
 NJS_TEXLIST *texlist
```

Parameter

***texlist**

NJS_TEXLIST structure pointer

Return Value

Success

1

Failure

-1

Function

Set current texture list to texlist

Notes

The texture list set herein will become the current texture list until the next call of njSetTexture. Texture functions, and such functions as njXXXXNum and njXXXXNumG, operate on the current texture list.

njSetTextureNum

Set current texture to texture number

Format

```
#include <Ninja.h>
 Sint32 njSetTextureNum(n);
 Uint32 n
```

Parameter

n

texture number n

Return Value

Success

1

Failure

-1

Function

Set texture number n in current texture list to current texture.

This will remain the current texture until the next calls of njSetTextureNum or njSetTextureNumG.

Notes

The assigned texture must be in texture memory.

njSetTextureNumG

Set current texture by global index number

Format

```
#include <Ninja.h>
 Sint32 njSetTextureNumG(globalIndex);
 Uint32 globalIndex
```

Parameter

globalIndex global index number

Return Value

Success

1

Failure

-1

Function

Set the current texture of global index number globalIndex to current texture. This will remain the current texture until the next calls of njSetTextureNum or njSetTextureNumG

Notes

The assigned texture must be in texture memory

njLoadCacheTextureNum

Load texture from cache memory to texture memory

Format

```
#include <Ninja.h>
Sint32 njLoadCacheTextureNum(n);
Uint32 n
```

Parameter

n

current texture list texture number

Return Value

Success

1

Failure

-1

Function

Load texture of texture number n from cache memory into texture memory

Notes

Current texture list must first be set using njSetTexture. Selected texture must be in cache memory.

njLoadCacheTextureNumG

Load texture by global index number from cache memory to texture memory

Format

```
#include <Ninja.h>
Sint32 njLoadCacheTextureNumG(globalIndex);
Uint32 globalIndex
```

Parameter

globalIndex

global index number

Return Value

Success

1

Failure

-1

Function

Load texture of global index number `globalIndex` from cache memory into texture memory

Notes

Selected texture must be in cache memory. Even if cache memory is released, textures in texture memory are not released.

njReleaseTextureAll

Release all texture memory

Format

```
#include <Ninja.h>
void njReleaseTextureAll(void);
```

Parameter

none

Return Value

none

Function

Release all texture memory

Notes

To use a texture again, that texture will have to be reloaded using njLoadTexture or related function.

njReleaseTexture

Release texture in texture list from texture memory

Format

```
#include <Ninja.h>
 Sint32 njReleaseTexture( *texlist );
 NJS_TEXLIST *texlist
```

Parameter

***texlist**

NJS_TEXLIST structure pointer

Return Value

Success

1

Failure

-1

Function

Release texture in texture list texlist from texture memory

Notes

In order to release a texture from texture memory, that texture must be released from any and all loaded texture lists in which the texture appears. Also, textures with the same global index number are considered the same texture.

njReleaseTextureNum

Release texture by texture number from texture memory

Format

```
#include <Ninja.h>
Sint32 njReleaseTextureNum(n);
Uint32 n
```

Parameter

n
texture number

Return Value

Success

1

Failure

-1

Function

Release current texture list texture of texture number n from texture memory

Notes

In order to release a texture from texture memory, that texture must be released from any and all loaded texture lists in which the texture appears. Also, textures with the same global index number are considered as the same texture.

njReleaseTextureNumG

Release texture by global index number from texture memory

Format

```
#include <Ninja.h>
 Sint32 njReleaseTextureNumG(globalIndex);
 Uint32 globalIndex
```

Parameter

globalIndex global index number

Return Value

Success

1

Failure

-1

Function

Release current texture list texture of global index number globalIndex from texture memory.

Notes

In order to release a texture from texture memory, that texture must be released from any and all loaded texture lists in which the texture appears. Also, textures with the same global index number are considered the same texture.

njReleaseCacheTextureAll

Release all cache memory

Format

```
#include <Ninja.h>
void njReleaseCacheTextureAll(void);
```

Parameter

none

Return Value

none

Function

Release all cache memory. Cache information area will not be released

Notes

Even if cache memory is released, textures in texture memory are not released.

njReleaseCacheTextureNum

Release texture by texture number from cache memory

Format

```
#include <Ninja.h>
 Sint32 njReleaseCacheTextureNum(n);
 Uint32 n
```

Parameter

n
texture number in current texture list

Return Value

Success

1

Failure

-1

Function

Release texture of texture number *n* from cache memory.

Notes

Current texture list must be set using `njSetTexture`. Selected texture must be loaded into cache memory. Even if cache memory is released, textures in texture memory are not.

njReleaseCacheTextureNumG

Release texture by global index number from cache memory

Format

```
#include <Ninja.h>
Sint32 njReleaseCacheTextureNumG(globalIndex);
Uint32 globalIndex
```

Parameter

globalIndex

global index number

Return Value

Success

1

Failure

-1

Function

Release texture of which global index number is globalIndex from cache memory.

Notes

Selected texture must be loaded into cache memory. Even if cache memory is released, textures in texture memory are not.

njGetTextureNumG

Get global index number of current texture

Format

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

Parameter

none

Return Value

Success

global index number from 0 to 0xFFFFFFFF

Failure

0xFFFFFFFF

Function

Get global index number of current texture

Notes

If current texture is not previously defined with `njSetTextureNum` or `njSetTextureNumG`, this function serves no purpose.

njCalcTexture

Calculate remaining texture memory

Format

```
#include <Ninja.h>
Uint32 njCalcTexture(Flag);
```

Parameter

 Uint32 flag NJD_TEXMEM_FREESIZE or
NJD_TEXMEM_MAXBLOCK
Specify NJD_TEXMEM_MAXSIZE

Return Value

Remaining amount of texture memory

Function

Calculate remaining texture memory

NJD_TEXMEM_FREESIZE	Texture memory free size
NJD_TEXMEM_MAXBLOCK	Texture memory maximum free block
NJD_TEXMEM_MAXSIZE	Total capacity of texture memory

Notes

njInitTexture must be called prior to this function.

njExitTexture

Quit texture usage

Format

```
#include <Ninja.h>
void njExitTexture(void);
```

Parameter

none

Return Value

none

Function

Quit texture usage. Also releases cache if that has not been done yet

Notes

Be sure to call this function when finished with textures.

njSetTexturePath

Set path of the directory which has texture

Format

```
#include <Ninja.h>
void njSetTexturePath(path);
```

Parameter

UInt8 *path Path to the directory

Return Value

none

Function

Set the path to the directory which has the texture. It is available for loading textures from files in njLoadTexture,njLoadTextureNum. The path set herein is available until it is changed.

Notes

This functions must be called prior to njLoadTexture and njLoadTextureNum.

njSetTextureInfo

Set information to the texture info structure.

Format

```
#include <Ninja.h>
void njSetTextureInfo(NJS_TEXINFO *,Uint16 *,Sint32,Sint32,Sint32)
```

Parameter

NJS_TEXINFO	*info	Texture Information (output)
Uint16	*tex	Pointer of memory texture
Sint32	Type	Texture type
Sint32	nWidth	Texture width
Sint32	nHeight	Texture length

Return Value

none

Functions

For memory textures, set texture information to the info of texture information structure.

Set color format and category code as Type. Set info which is set herein to addr of NjSetTexturename.

Note

See sample program for the way to use.

Color format

NJD_TEXFMT_ARGB_1555	
NJD_TEXFMT_RGB_565	
NJD_TEXFMT_ARGB_4444	
NJD_TEXFMT_YUV_422	Not available
NJD_TEXFMT_BUMP	Not available

Category code

NJD_TEXFMT_TWIDDLED	
NJD_TEXFMT_TWIDDLED_MM	
NJD_TEXFMT_VQ	Not available
NJD_TEXFMT_VQ_MM	Not available
NJD_TEXFMT_PALETTIZE4	Not available
NJD_TEXFMT_PALETTIZE4_MM	Not available
NJD_TEXFMT_PALETTIZE8	Not available
NJD_TEXFMT_PALETTIZE8_MM	Not available
NJD_TEXFMT_RECTANGLE	
NJD_TEXFMT_STRIDE	

njSetTextureName

Set data to texture name structure.

Format

```
#include <Ninja.h>
void njSetTextureName(NJS_TEXNAME *,void *,Uint,Uint32)
```

Parameter

NJS_TEXNAME	*texname	Texture name structure (output)
void	*addr	File name or pointer for NJD_TEXINFO structure
	Uint32	globalIndex Global index
	Uint32	attr Texture attribute

Return value

None

Functions

Set filenames to addr to load textures from files.

Specify NJD_TEXATTR_TYPE_FILE to attr. For textures of PVR format, in case of using global index in files, set 0xFFFFFFFF to globalIndex. If 0 to 0xFFFFFFFF is set to globalIndex, the number has priority over global index in PVR files. In case of memory textures, the pointer of NJS_TEXINFO structure which is set in njSetTextureInfo is set to addr. Set NJD_TEXATTR_TYPE_MEMORY to attr and set globalIndex to global index.

Notes

See sample program for the information on how to use this function.

njReloadTextureNum (New Function)

Overview

Reloads a texture by texture number

Syntax

```
#include <Ninja.h>
Shint32 njReloadTextureNum(n, texaddr, attr, lod);
```

Parameters

Unit32	Current texture list texture number
Void*texaddr	Filename or texture memory address
Unit32attr	Texture attribute
Unit32lod	Mip-map level

Return Value

Success

1

Failure

-1

Purpose

Reloads texture number *n* in the current texture list. The reloaded texture is the same as that loaded before. Set *attr* to `NJD_TEXATTR_TYPE_FILE` to load the texture from a file, or to `NJD_TEXATTR_TYPE_MEMORY` to load the texture from memory.

For a mip-map texture, reload *lod* with the corresponding mip-map level. For example, setting *lod* to 128 reloads only the 128 \diamond 128 texture level. To reload all mip-map texture levels, set *lod* to 0. When loading from memory, reload the *lod* level from the address specified by *texaddr*.

Remarks

For the texture memory case, specify the head of the texture that was set by *lod*.

njReloadTextureNumG (New Function)

Overview

Reload the global index number texture.

Syntax

```
#include <Ninja.h>
Shint32 njReloadTextureNumG(globalIndex, texaddr, attr, lod);
```

Parameters

Unit32n	Global index texture number
Void*texaddr	Filename or texture memory address
Unit32attr	Texture attribute
Unit32lod	Mip map level

Return Value

Success

1

Failure

-1

Purpose

Reloads the texture of the global index number `globalindex`. The reloaded texture is the same as that loaded before. Set `attr` to `NJD_TEXATTR_TYPE_FILE` to load the texture from a file, or to `NJD_TEXATTR_TYPE_MEMORY` to load the texture from memory.

For a mip-map texture, reload `lod` with the corresponding mip-map level. For example, setting `lod` to 128 reloads only the 128 \diamond 128 texture level. To reload all mip-map texture levels, set `lod` to 0. When loading from memory, reload the `lod` level from the address specified by `texaddr`.

Remarks

For the texture memory case, specify the head of the texture that was set by `lod`.

njSetRenderWidth (New Function)

Overview

Sets a Stride value.

Syntax

```
#include <Ninja.h>
void njSetRenderWidth(nWidth);
```

Parameter

Uint32 nWidthStride value

Return Value

None

Purpose

Sets a Stride value when using a Stride texture format. When specifying a Stride texture with a render texture, if the texture is smaller than the rendering area, set the width of the texture. Otherwise, if the rendering area is smaller than the texture, set the width of the rendering area. Acceptable values are multiples of 32 from 32 to 992.

Remarks

NjFrameBufferBmp (New Function)

Overview

Make a frame buffer into a bitmap.

Syntax

```
#include <Ninja.h>
void njFrameBufferBmp(filename);
```

Parameter

Unit8*filename

File name

Return Value

None

Purpose

Makes a frame buffer into a 24-bit BMP. Currently only frame 0 can be a texture, so when frame 0 uses this function during drawing, the partially rendered image appears. (Planned to be changed later)

Remarks

In the future, the displayed frame will be modified to a BMP. Use for debugging.

Deleted functions

NjInitCache Texture
njLoadTextureNumG

Texture Structures

NJS_TEXSURFACE

```
typedef struct{
    Uint32      Type;           /**/
    Uint32      BitDepth;      /**/
    Uint32      PixelFormat;   /**/
    Uint32      nWidth;        /**/
    Uint32      nHeight;       /**/
    Uint32      TextureSize;   /**/
    Uint32      fSurfaceFlags; /**/
    Uint32      *pSurface;     /**/
}NJS_TEXSURFACE;
```

NJS_TEXINFO

```
typedef struct{
    void*      texaddr;        /* texture buffer address */
    NJS_TEXSURFACE texsurface /* texture surface address */
} NJS_TEXINFO;
```

NJS_TEXNAME

```
typedef struct {
    void      *filename;      /* texture filename strings */
    Uint32    globalIndex;    /* global unique texture ID */
    Uint16    attr;           /* texture attribute */
    Uint8     width;          /* texture width */
    Uint8     height;         /* texture height */
    Uint32    texaddr;        /* texture memory address cache */
} NJS_TEXNAME;
```

NJS_TEXLIST

```
typedef struct {
    NJS_TEXNAME    textures;    /* texture array          */
    Uint32         nbTexture;   /* texture count         */
} NJS_TEXLIST;
```

NJS_TEXMEMLIST

```
typedef struct {
    Uint32         globalIndex; /* global unique texture ID */
    Uint32         texaddr;    /* texture memory address cache */
    NJS_TEXINFO    texinfo;   /* texinfo                 */
    Uint16         count;     /* texture count           */
    Uint16         dummy;
} NJS_TEXMEMLIST;
```

Texture Definitions**Used with nWidth, nHeight**

```
#define NJD_TEXSIZE_1      1
#define NJD_TEXSIZE_2      2
#define NJD_TEXSIZE_4      4
#define NJD_TEXSIZE_8      8
#define NJD_TEXSIZE_16     16
#define NJD_TEXSIZE_32     32
#define NJD_TEXSIZE_64     64
#define NJD_TEXSIZE_128    128
#define NJD_TEXSIZE_256    256
#define NJD_TEXSIZE_512    512
#define NJD_TEXSIZE_1024   1024
```

Used with attr

```
Texture load source
#define NJD_TEXATTR_TYPE_FILE      0
#define NJD_TEXATTR_TYPE_MEMORY    BIT_30 Load from memory

Texture load source
#define NJD_TEXATTR_TYPE_FRAMEBUFFER BIT_28 Load from frame buffer
#define NJD_TEXATTR_CACHE          BIT_31 Load into cache
#define NJD_TEXATTR_BOTH           BIT_29 Load into both cache and
#define NJD_TEXATTR_MASK          0xF0000000

#define NJD_TEXATTR_READAREA_MASK  (BIT_31|BIT_29)
#define NJD_TEXATTR_READTYPE_MASK (BIT_30|BIT_28)

#define NJD_TEXATTR_GLOBALINDEX    BIT_23 Change
#define NJD_TEXATTR_AUTOMIPMAP     BIT_22 Complies with the next period
#define NJD_TEXATTR_AUTODITHER    BIT_21 Complies with the next period
#define NJD_TEXATTR_MASK          0xFFFF0000
```

Used with Type, color format

```
#define NJD_TEXFMT_ARGB_1555      (0x00)
#define NJD_TEXFMT_RGB_565       (0x01)
#define NJD_TEXFMT_ARGB_4444     (0x02)
#define NJD_TEXFMT_YUV_422       (0x03)      Not available
#define NJD_TEXFMT_BUMP          (0x04)      Not available
#define NJD_TEXFMT_COLOR_MASK    (0xFF)
```

Category code

```
#define NJD_TEXFMT_TWIDDLED              (0x0100)
#define NJD_TEXFMT_TWIDDLED_MM          (0x0200)
#define NJD_TEXFMT_VQ                    (0x0300)      Not available
#define NJD_TEXFMT_VQ_MM                 (0x0400)      Not available
#define NJD_TEXFMT_PALETTIZE4            (0x0500)      Not available
#define NJD_TEXFMT_PALETTIZE4_MM        (0x0600)      Not available
#define NJD_TEXFMT_PALETTIZE8            (0x0700)      Not available
#define NJD_TEXFMT_PALETTIZE8_MM        (0x0800)      Not available
#define NJD_TEXFMT_RECTANGLE              (0x0900)
#define NJD_TEXFMT_STRIDE                  (0x0B00)
#define NJD_TEXFMT_TWIDDLED_RECTANGLE    (0x0D00)      Not available
#define NJD_TEXFMT_ABGR                    (0x0E00)      Not available
#define NJD_TEXFMT_ABGR_MM                 (0x0F00)      Not available
#define NJD_TEXFMT_TYPE_MASK              (0xFF00)
```

Texture error code (new addition)

```
#define NJD_TEXERR_OTHER              (1)  //Other errors
#define NJD_TEXERR_FILEOPEN           (2)  //File open error
#define NJD_TEXERR_EXTND               (3)  //Extention error
#define NJD_TEXERR_HEADER              (4)  //Header error
#define NJD_TEXERR_FILELOAD            (5)  //File load error
#define NJD_TEXERR_SURFACE              (6)  //Surface creation error
#define NJD_TEXERR_MAINMEMORY          (7)  //Main memory malloc error
#define NJD_TEXERR_TEXMEMLOAD          (8)  //Texture memory load error
#define NJD_TEXERR_GLOBALINDEX         (9)  //Global Index Error
```

Acquire texture memory size (used with njCalcTexture)

```
#define NJD_TEXMEM_FREESIZE            (0x00000000)
#define NJD_TEXMEM_MAXBLOCK            (0x00000001)
#define NJD_TEXMEM_MAXSIZE              (0x00000002)
```

Sample Program

Overview

This chapter contains simple sample programs illustrating the following examples:

- Ex. 1: Display of a PVR texture file.
- Ex. 2: Load a texture from memory and display it.
- Ex. 3: Load file from cache and display texture.

Sample

Ex. 1 : Display of PVR texture file

```

1:#include <Ninjawin.h>
2:
3:NJS_TEXNAME texname[2];
4:
5:NJS_TEXLIST texlist = {texname,2};
6:NJS_TEXMEMLIST texmemlist[2]; /*Reserve texture information area for 2 textures*/
7:NJS_POINT2COL p[4];
8:
9: void njUserInit(void)
10:{
11:     njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 );
12:     /* Set two textures */
13:     njSetTextureName(&texname[0],"file0.pvr",0,NJD_TEXATTR_TYPE_FILE);
14:     njSetTextureName(&texname[1],"file1.pvr",1,NJD_TEXATTR_TYPE_FILE);
15:     njInitTexture(texmemlist,2);
16:     njLoadTexture(&texlist); /* Load textures */
17:     njSetTexture(&texlist); /* Assignt texlist to Current texture list */
18:     /* Assign current texture to texture 0 of texlist*/
19:     njSetTextureNum(0);
20:
21:     /* Polygon data input */
22:     p[0].x = 100; p[0].y = 100;
23:     p[1].x = 200; p[1].y = 100;
24:     p[2].x = 200; p[2].y = 200;
25:     p[3].x = 100; p[3].y = 200;
26:     p[0].col.tex.u = 0; p[0].col.tex.v = 0;
27:     p[1].col.tex.u = 255; p[1].col.tex.v = 0;
28:     p[2].col.tex.u = 255; p[2].col.tex.v = 255;
29:     p[3].col.tex.u = 0; p[3].col.tex.v = 255;
30:}
31:Sint32 njUserMain(void)
32{
33:     /* Draw polygon of texture */
34:     njDrawPolygon2D(p,4,-100.f,NJD_FILL|NJD_USE_TEXTURE);

```

```
35:     return NJD_USER_CONTINUE;
36: }
37:
38: void njUserExit(void)
39: {
40:     njExitTexture();
41:     njExitSystem();
42: }
43: }
```

Ex. 2: Load a texture from memory and display it.

```
1: #include <Ninjawin.h>
2:
3: extern Uint16 Image[]; /* Assume there is mipmap data over 256 in other file */
4:
5: NJS_TEXINFO Info;
6: NJS_TEXNAME texname[2];
7:
8: NJS_TEXLIST texlist = {texname, 2};
9: NJS_TEXMEMLIST texmemlist[2]; /* Reserve texture information area for 2 textures */
10: NJS_POINT2COL p[4];
11:
12: void njUserInit(void)
13: {
14:     njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 )
15:     /* Set 2 textures*/
16:
17:     jSetTextureInfo(&Info, Image, NJD_TEXFMT_TWIDDLED|NJD_TEXFMT_ARGB_1555, 256, 256);
18:     njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE);
19:     njSetTextureName(&texname[1], &Info, 1, NJD_TEXATTR_TYPE_MEMORY);
20:     njInitTexture(texmemlist, 2);
21:     njLoadTexture(&texlist); /* Load texture */
22:     njSetTexture(&texlist); /* Assign texlist to current texture list */
23:     /* Assing texture 1 of texlis to current texture */
24:     njSetTextureNum(1);
25:
26:     /* Input plygon data */
27:     p[0].x = 100; p[0].y = 100;
28:     p[1].x = 200; p[1].y = 100;
29:     p[2].x = 200; p[2].y = 200;
30:     p[3].x = 100; p[3].y = 200;
31:     p[0].col.tex.u = 0; p[0].col.tex.v = 0;
32:     p[1].col.tex.u = 255; p[1].col.tex.v = 0;
33:     p[2].col.tex.u = 255; p[2].col.tex.v = 255;
34:     p[3].col.tex.u = 0; p[3].col.tex.v = 255;
35: }
36: Sint32 njUserMain(void)
37: {
38:     /* Draw polygon of texture */
39:     njDrawPolygon2D(p, 4, -100.f, NJD_FILL|NJD_USE_TEXTURE);
```

```
40:     return NJD_USER_CONTINUE;
41: }
42:
43: void njUserExit(void)
44: {
45:     njExitTexture();
46:     njExitSystem();
47: }
```

Ex. 3: Load file from cache and display texture

```
1: #include <Ninjawin.h>
2:
3: NJS_TEXNAME texname[2];
4: 5: NJS_TEXLIST texlist = {texname, 2};
6: NJS_TEXMEMLIST texmemlist[2]; /* Reserve texture information area for 2 textures */
7: NJS_POINT2COL p[4];
8: 12: void njUserInit(void):
9: {
10:     njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1 )
11:     njInitTexture(texmemlist, 2);
12:     /* Set 2 textures */
13:     njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE |
14:                     NJD_TEXATTR_CACHE | NJD_TEXATTR_GLOBALINDEX);
15:     njSetTextureName(&texname[1], "file1.pvr", 1, NJD_TEXATTR_TYPE_MEMORY |
16:                     NJD_TEXATTR_CACHE | NJD_TEXATTR_GLOBALINDEX);
17:     njLoadTexture(&texlist); /* Load textures */
18:     njSetTexture(&texlist); /* Specify texlist to current texture */
19:     njLoadCacheTextureNum(0); /* Load texture of number 0 from cache */
20:     njLoadCacheTextureNum(1); /* Load texture of number 1 from cache */
21:     /* Assign texture 0 of texlist to current texture */
22:     njSetTextureNum(0);
23:
24:     /* Polygon data input */
25:     p[0].x = 100; p[0].y = 100;
26:     p[1].x = 200; p[1].y = 100;
27:     p[2].x = 200; p[2].y = 200;
28:     p[3].x = 100; p[3].y = 200;
29:     p[0].col.tex.u = 0; p[0].col.tex.v = 0;
30:     p[1].col.tex.u = 255; p[1].col.tex.v = 0;
31:     p[2].col.tex.u = 255; p[2].col.tex.v = 255;
32:     p[3].col.tex.u = 0; p[3].col.tex.v = 255;
33: }
34:
35: Sint32 njUserMain(void)
36: {
37:     /* Draw texture polygon */
38:     njDrawPolygon2D(p, 4, -100.f, NJD_FILL | NJD_USE_TEXTURE);
39:     return NJD_USER_CONTINUE;
40: }
```

```
41:
42: void njUserExit(void)
43: {
44:     njExitTexture();
45:     njExitSystem();
46: }
```



03. NINJA LIGHT

How to set LIGHT

>> void njCreateLight(NJS_LIGHT*, Int)

Parameter: NJS_LIGHT *ptr
 Int lsrc

Description: This function defines the kind of light source lsrc and registers Light ptr newly.

Return Value: None

Remarks: None

>> void njDeleteLight(NJS_LIGHT*)

Parameter: NJS_LIGHT *ptr

Description: This function deletes created Light ptr.

Return Value: None

Remarks: None

>> void njLightOff(NJS_LIGHT*)

Parameter: NJS_LIGHT *ptr

Description: This function does not reflect set Light ptr.

Return Value: None

Remarks: You can use macro.

>> void njLightOn(NJS_LIGHT*)

Parameter: NJS_LIGHT *ptr
Description: This function reflects set Light ptr in the model.
Return Value: None
Remarks: You can use macro.

>> void njMultiLightMatrix(NJS_LIGHT*, NJS_MATRIX*)

Parameter: NJS_LIGHT *ptr
NJS_MATRIX *m
Description: This function multiples Light matrix registered by njCreateLight and matrix m.
Return Value: None
Remarks: The scale factor should not be included in Matrix m.

>> void njSetLight(NJS_LIGHT*)

Parameter: NJS_LIGHT *ptr
Description: This function registers Light ptr newly which is already defined by tools.
Return Value: None
Remarks: None

>> void njSetLightAlpha(NJS_LIGHT*, Float)

Parameter: NJS_LIGHT *ptr
Float alpha
Description: This function sets alpha value for the light registered by njCreateLight.
Return Value: None
Remarks: TBS

>> void njSetLightAngle(NJS_LIGHT*, NJS_Angle, NJS_Angle)

Parameter: NJS_LIGHT *ptr
NJS_Angle iang
NJS_Angle oang
Description: This function sets limit angle value for the light registered by njCreateLight.
Return Value: None
Remarks: Only spot light is used (for now)

>> void njSetLightColor(NJS_LIGHT*, Float, Float, Float)

Parameter: NJS_LIGHT *ptr
 Float red
 Float green
 Float blue

Description: This function sets RGB value for the light registered by njCreateLight.

Return Value: None

Remarks: None

>> void njSetLightDirection(NJS_LIGHT*, Float, Float, Float)

Parameter: NJS_LIGHT *ptr
 Float dx
 Float dy
 Float dz

Description: This function sets the light source direction for the light registered by njCreateLight.

Return Value: None

Remarks: Only parallel light source and spotlight are used.(for now)

>> void njSetLightIntensity(NJS_LIGHT*, Float, Float, Float)

Parameter: NJS_LIGHT *ptr
 Float spc
 Float dif
 Float amb

Description: This function sets the intensity of light registered by njCreateLight.

Return Value: None

Remarks: None

>> void njSetLightLocation(NJS_LIGHT*, Float, Float, Float)

Parameter: NJS_LIGHT *ptr
 Float px
 Float py
 Float pz

Description: This function sets the location of light registered by njCreateLight.

Return Value: None

Remarks: Only point light source and spotlight is used. (for now)

>> void njSetLightRange(NJS_LIGHT*, Float, Float)

Parameter: NJS_LIGHT *ptr
 Float nrang
 Float frang

Description: This function sets limit range value for the light registered by njCreateLight.

Return Value: None

Remarks: Only point light source and spotlight is used. (for now)

>> void njSetUserLight(NJS_LIGHT*, NJF_LIGHT_FUNC*)

Parameter: NJS_LIGHT *ptr
 NJF_LIGHT_FUNC func

Description: This function sets user setting light function func for Light ptr.

Return Value: None

Remarks: None

>> void njUnitLightMatrix(NJS_LIGHT*)

Parameter: NJS_LIGHT *ptr

Description: This function sets light matrix registered by njCreateLight as unit matrix

Return Value: None

Remarks: None

>> void njTranslateLightV(NJS_LIGHT*, NJS_VECTOR*)

Parameter: NJS_LIGHT *ptr
 NJS_VECOTR *vctr

Description: This function translates light matrix registered by njCreateLight in the direction of vector vctr.

Return Value: None

Remarks: None

>> void njTranslateLight(NJS_LIGHT*, Float, Float, Float)

Parameter: NJS_LIGHT *ptr
 Float tx
 Float ty
 Float tz

Description: This function translates light matrix registered by njCreateLight in the direction of (tx, ty, tz).

Return Value: None

Remarks: None

>> void njRotateLightX(NJS_LIGHT*, NJS_Angle)

Parameter: NJS_LIGHT *ptr
 NJS_Angle ang

Description: This function rotates light matrix registered by njCreateLight around X axis at ang angle.

Return Value: None

Remarks: None

>> void njRotateLightXYZ(NJS_LIGHT*, NJS_Angle, NJS_Angle,

NJS_Angle)

Parameter: NJS_LIGHT *ptr
 NJS_Angle xang
 NJS_Angle yang
 NJS_Angle zang

Description: This function rotates light matrix resgistered by njCreateLight around XYZ axis.

Return Value: None

Remarks: None

>> void njRotateLightY(NJS_LIGHT*, NJS_Angle)

Parameter: NJS_LIGHT *ptr
 NJS_Angle ang

Description: This function rotates light matrix registered by njCreateLight around Y axis at ang angle.

Return Value: None

Remarks: None

>> void njRotateLightZ(NJS_LIGHT*, NJS_Angle)

Parameter: NJS_LIGHT *ptr
 NJS_Angle ang

Description: This function rotates light matrix registered by njCreateLight around Z axis at ang angle.

Return Value: None

Remarks: None

Macro

```
NJS_LIGHT * l

#define NJM_LIGHT_INIT_VECTOR(l) l->vctr      (The vector of initial light)
#define NJM_LIGHT_INIT_POINT(l) l->pnt       (The point of initial light)
#define NJM_LIGHT_MATRIX(l)    l->mtrx       (The matlrix of light)
#define NJM_LIGHT_VECTOR(l)    (l->ltcal).lvctr (The present vector of light)
#define NJM_LIGHT_POINT(l)    (l->ltcal).lpnt (The present point of light)
#define NJM_LIGHT_AMB(l)      (l->ltcal).amb  (The intensity of ambient light)
#define NJM_LIGHT_DIF(l)      (l->ltcal).dif  (The intensity of diffused light)
#define NJM_LIGHT_SPC(l)      (l->ltcal).spc  (The intensity of specular light)
#define NJM_LIGHT_EXP(l)      (l->ltcal).exp  (The Index number:exponent for specular)
#define NJM_LIGHT_COLOR(l)    (l->attr).argb  (The color of light)
```

How to Use

The calculation of light source is based on the light structure which describes necessary light information such as location, direction, color and kind of light. Light structure is set by 2 kinds of light functions.

Light function `njCreateLight` is generally used, and creates new light structure based on the kinds of light sources which are defined by arguments. You can add more detailed light source information by using `njSetLight` functions...

The other way to set the light structure is to use `njSetLight`. This method is used when the light source information has already been set. Please note that it registers light source (=light structure) only.

The registered light source is reflected on the model by default. If you want to stop calculation of the light source for a certain model, you must set `njLightOff` before drawing the model.

Please note that `njLightOff` and `njLightOn` keep current status. Please refer to Reference for more detailed information about functions, arguments, structures, etc.

Example1

Sets Light `lt1` as spotlight and Light`lt2` as ambient light + point light source (Lambert model).

```
#include <ninja.h>
.....
// Declare Light.
NJS_LIGHT lt1, lt2;
.....
// This is initial routine
/* Initialize and register Light.*/
njCreateLight(&lt1,  NJD_SPOT_LIGHT);
njSetLightAngle(&lt1,  DegToAngle(30.f),  DegToAngle(60.f));
njSetLightRange(&lt1,  1000.f,  1500.f);
njSetLightLocation(&lt1,  0.f,  10.f,  15.f);
njSetLightDirection(&lt1,  0.f,  1.f,  0.f);
njSetLightColor(&lt1,  1.f,  0.f,  0.f);
njCreateLight(&lt2,  NJD_LAMBERTIAN_POINT);
/*Set various Light property.*/
njSetLightColor(&lt2,  0.5f,  0.5f,  0.5f);
njSetLightIntensity(&lt2,  0.f,  1.f,  1.f); // Default intensity is (1.f, 1.f, 1.f).
```

```
njSetLightRange(&lt2, 100.f, 1500.f);
.....
// Drawing routine is as follows.
while(-1)
{
.....

    /* Reflect Light lt1, lt2 on the model. */
    njDrawModel(...);

.....

    /* Remove Light lt2. */
    njLightOff(lt2);

    /* Reflect Light lt1 on the model. */
    njDrawModel(...);

    /* Reflect Light lt2 on the model. */
    njLightOn(lt2);

.....
}
```

Example2

Changes spot light color of Light lt1 by branch processing and add parallel light source lt3 newly.

```
.....
/* Change color of Light lt1. */
njSetLightColor(&lt1, 0.f, 1.f, 1.f);
/* Initialize and register Light. */
njCreateLight(&lt3,NJD _ DIRECTIONAL_LIGHT);
/* Set various kinds of Light property. */
njSetLightDirection(&lt3, 1.f, 0.f, 0.f);
njSetLightIntensity(&lt3, 0.f, 1.f, 1.f);
.....
// Drawing routine is as follows.
while(-1)
{
.....

    /* Reflect Light lt1, lt2, lt3on the model. */
    njDrawModel(...);

.....
}
```

Example3

```
Sets user functions for Light lt.
//Set up user functions .(The arguments of functions is as follows.)
void
userfunc(NJS_ARGB* argb, NJS_POINT3* pnt, NJS_VECTOR* nml, NJS_LIGHT_PTR light)
{
    .....
    // Internal product of polygon normal vector and direction of light
    deg = - nml->x * NJM_LIGHT_VECTOR(light).x
        - nml->y * NJM_LIGHT_VECTOR(light).y
        - nml->z * NJM_LIGHT_VECTOR(light).z;
    .....
    argb->a = deg * NJM_LIGHT_DIF(light).a;
    argb->r = deg * NJM_LIGHT_DIF(light).r;
    argb->g = deg * NJM_LIGHT_DIF(light).g;
    argb->b = deg * NJM_LIGHT_DIF(light).b;
}
//Main routine (omit some part)
.....
njCreateLight(&lt, NJD_USER_LIGHT);
.....
/*Set User function userfunc for Light lt*/
njSetUserLight (&lt, userfunc);
/*Color setting for Light lt*/
njSetLightColor(&lt, 0.f, 1.f, 1.f);
.....
// Drawing routine is as follows.
while(-1)
{
    .....

    /* Reflect Light lt on the model. */
    DrawModel(...);

    .....
}
```

LIGHTstructure Specification

Though users do not have to use Light structure directly, we will show you the specification below.

Ninja Softimage	[NJS_MATERIALstructure]	[NJS_LIGHT_ATTRstructure]
[specular]	argb or rgb	intensity_spec
[diffuse]	argb or rgb	intensity_diff
[ambient]	(argb or rgb :pending)	intensity_amb
[exponent]	exp	None

specular:highlight	Softimage sets 0 to 1 for normal RGB value.
diffuse: Normal light	Softimage sets 0 to 1 for normal RGB value.
ambient:Ambient light	Softimage sets 0 to 1 for normal RGB value.
exponent:Exponent for highlight	Softimage set 0 to 300 for normal RGB value.

(We will not support HSV at Ninja Library.)

The members of NJS_LIGHT structure

```
struct {
    BOOL          stat;      (Status:Use/Not use of Lightsource)
    NJS_POINT3    pnt;      (Point of light source)
    NJS_VECTOR     vctr;    (Light source unit vector)
    NJS_MATRIX    mtrx;    (Light source matrix)
    NJS_LIGHT_ATTR attr;    (Attribute structure)
    NJS_LIGHT_CAL ltcal;    (Light calculation structure)
} NJS_LIGHT;
```

<stat>

```
#define  NJD_LIGHT_ON      Reflects light
#define  NJD_LIGHT_OFF    Do not reflect light
```

The members of NJS_LIGHT_ATTR structure

```
struct {
    Int          lsrc;      (Kind of light source)
    Float        ispc;      (Intensity of specular light:0 to 1)
    Float        idif;      (Intensity of diffusion:0 to 1)
    Float        iamb;      (Intensity of ambience:0 to 1)
    Float        nrang;     (Range of maximum light intensity:Limit value in front)
    Float        frang;     (Range for cutting off light intensity:Limit value of back)
    void*        func;      (Pointer of callback function)
    Angle        iang;      (Angle of maximum light intensity:Inside limit angle)
    Angle        oang;      (Range for cutting off light intensity:Outside limit angle)
    NJS_ARGB     argb;      (Color of light)
} NJS_LIGHT_ATTR
```

</src>

The kinds of light source

```
#define NJD_SPOT_LIGHT          Spot light
#define NJD_DIR_LIGHT          Parallel light source
#define NJD_POINT_LIGHT        point light source
#define NJD_AMBIENT            Ambience
#define NJD_SPEC_DIR           Parallel light source highlight
#define NJD_SPEC_POINT         Point light source highlight
#define NJD_LAMBERTIAN_DIR     Parallel light source lambert
#define NJD_LAMBERTIAN_POINT   Point light source lambert
#define NJD_PHONG_DIR          Parallel light source phong
#define NJD_PHONG_POINT        Point light source phong
#define NJD_USER_LIGHT         User set light
#define NJD_BLOCK_LIGHT        Block light
```

<ispc, idif, iamb>

The balance of light (is calculated as follows).

$\text{SPECULAR}(R,G,B) \times \text{ispc} + \text{DIFFUSE}(R,G,B) \times \text{idif} + \text{AMBIENT}(R,G,B) \times \text{iamb}$

But, top (bottom) of the limit is clamped.

<near, far>

The effective range of light which is defined by NJD_POINT_LIGHT (point light source), NJD_SPOT_LIGHT (spot light).

nrang The limit value of range which light is upper limit value. Default value:1.f

frang The limit value of range for calculation of the light. Default value:65535.f

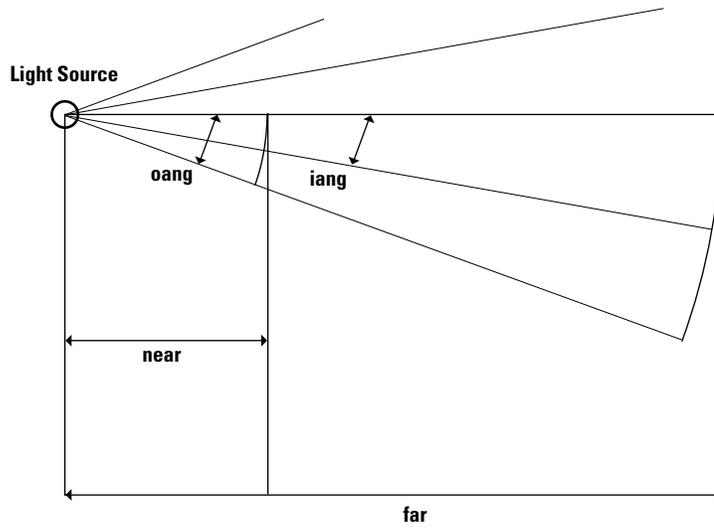
<iang, oang>

The effective range of light which is defined by NJD_SPOT_LIGHT (spotlight) etc.

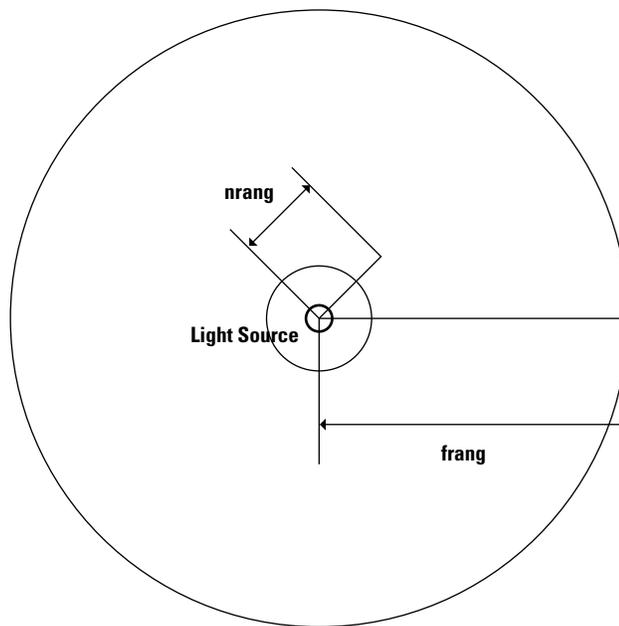
iang The limit value of angle which light is upper limit value. Default value:(DEG)10.f

oang The limit value of angle for calculation of the light. Default value:(DEG)30.f

(Example:Spot light)



(Example:point light source)



The members of NJS_LIGHT_CAL structure

```
struct
{
    Float      ratten;      (Attenuation rate: It is used by block light )
    Float      ipd;        (Inner Product:It is used by block light )
    Float      nrr;        (Limit judgement value of the light source, near:nrang * nrang)
    Float      frr;        (Limit judgement value of the light source, far:frang *
frang)
    Float      cosi;       (Limit judgement value of the light source, internal:cos *
cos)
    Float      cose;       (Limit judgement value of the light source, external:cos *
os)
    Float      ideo;       (Division judgement value of the light source, inter )
    Float      odev;       (Division judgement value of the light source, outer )
    Float      rate;       (Attenuaion ratio of light source - for spot light)
    Float      intns;      (Intensity of light source, 0 to 1)
    Int        exp;        (Diffusion exponent of light source)
    Int        reserve;    (reserve)
    NJS_POINT3 lpnt;       (Point of light source)
    NJS_VECTOR lvctr;      (Directional vector of light source)
    NJS_VECTOR lmvctr;     (Directional vector of light source: It is used by block light)
    NJS_ARGB   atten;      (intns * argb(Color of light source))
    NJS_ARGB   amb;        (iamb*atten)
    NJS_ARGB   dif;        (idif*atten)
    NJS_ARGB   spc;        (ispc*atten)
} NJS_LIGHT_CAL;

<exp>
```

This parameter gives glossiness. It is used in material structure. (This parameter is related to the “specular exponent” used in many lighting models.)



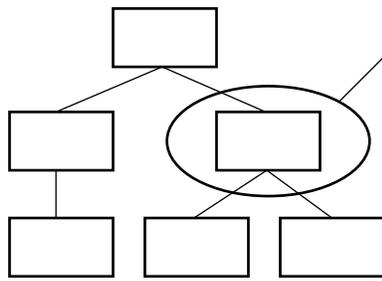
04. Ninja Model and Motion

Model Structures

See following page for diagram.

Diagram of Structure

Object Tree



NJS_OBJECT structure

Gives parent-child hierarchy of model.

```

typedef struct obj {
    Uint32 evalflags; /* Evaluation method optimization */
    NJS_MODEL *model; /* Model structure */
    Float pos[3]; /* Parallel motion */
    Angle ang[3]; /* Rotation */
    Float scl[3]; /* Scale */
    struct obj *child; /* Child pointer */
    struct obj *sibling; /* Sibling pointer */
} NJS_OBJECT;
    
```

NJS_MODEL structure

Gives vertices, (contiguous) polygons, and material data.

```

typedef struct {
    NJS_POINT3 *points; /* Vertex list */
    NJS_VECTOR *normals; /* Normal line vector list */
    Uint32 nbPoint; /* Number of points */
    NJS_MESHSET *meshsets; /* Polygon list */
    NJS_MATERIAL *mats; /* Material lists */
    Uint16 nbMeshset; /* Number of mesh lists */
    Uint16 nbMat; /* Number of mats */
    NJS_POINT3 center; /* Model center */
    Float r; /* Radius of circumscribed sphere */
} NJS_MODEL;
    
```

```

typedef struct {
    Float x, y, z;
} NJS_POINT3, NJS_VECTOR;
    
```

```

typedef struct {
    NJS_COLOR diffuse;
    NJS_COLOR specular;
    Float exponent;
    Uint32 attr_texId;
    Uint32 attrflags;
} NJS_MATERIAL;
    
```

NJS_OBJECT structure

TRIMESH and polygon list by material and type

```

typedef struct {
    Uint16 type_matId; /* Type and material ID */
    Uint16 nbMesh; /* Total number of polygons */
    Sint16 *meshes; /* Polygon list */
    Uint32 *attrs; /* Polygon attributes */
    NJS_VECTOR *normals; /* Polygon normal line list */
    NJS_COLOR *vertcolor; /* Vector color list */
    NJS_COLOR *vertuv; /* Vector UV list */
} NJS_MESHSET;
    
```

```

typedef union {
    Uint32 color;
    struct { Sint16 u, v; } tex;
    struct { Sint8 b,g,r,a; } argb;
} NJS_COLOR;
    
```

→ If "attrs", "normals", "vertcolor", and "vertuv" for the polygon vertices are not needed, these variables are NULL.

- The two upper bits give the data type. (There are four types.)
- List of triangular polygons only
- List of quadrilateral polygons only
- List of N-sided polygons
- Contiguous polygons (TRIMESH)

Description of Structures

Float, Angle

```
typedef float Float    /* Floating-point operation type */
typedef Sint32 Angle  /* Angle of rotation */
```

- For angles, 0x000 to 0xFFFF correspond to 0 to 360 degrees.

Color structure

```
typedef union {
    Uint32 color;          /* Long access */
    struct {
        Sint16 u;          /* Texture u value */
        Sint16 v;          /* Texture v value */
    } tex;                 /* Texture access */
    struct {
        Uint8 b;           /* b value */
        Uint8 g;           /* g value */
        Uint8 r;           /* r value */
        Uint8 a;           /* Alpha blend value */
    } argb;                /* argb access */
} NJS_COLOR;
```

- This structure stores colors and texture UVs. This structure uses a union.
- This tool sets the data from “color” and accesses the library from “tex” and “argb”.

Object structure

```
typedef struct obj {
    Uint32      evalflags; /* Evaluation method optimization flag */
    NJS_MODEL *model;     /* Model structure pointer */
    Float       pos[3];   /* Parallel motion */
    Angle       ang[3];   /* Rotation */
    Float       scl[3];   /* Scale */
    struct obj *child;    /* Child object pointer */
    struct obj *sibling;  /* Sibling object pointer */
} NJS_OBJECT;
```

- This structure gives the parent-child hierarchy of a model.
- Polygons and TRIMESHes (contiguous polygons) are set in “model”.

Explanation of evalflags

```
#define NJD_EVAL_UNIT_POS      BIT_0 /* Motion can be ignored */
#define NJD_EVAL_UNIT_ANG     BIT_1 /* Rotation can be ignored */
#define NJD_EVAL_UNIT_SCL     BIT_2 /* Scale can be ignored */
#define NJD_EVAL_HIDE        BIT_3 /* Do not draw model */
#define NJD_EVAL_BREAK       BIT_4 /* Break child trace */
#define NJD_EVAL_ZXY_ANG     BIT_5
/* Specification for evaluation of rotation that can be expected by LightWave3D */
#define NJD_EVAL_MASK        0x3f
/* Mask for extracting the above bits */
```

These flags are set automatically by the converter.

- NJD_EVAL_UNIT_POS is set when the parallel motion amount is “0”. The parallel motion matrix calculation is omitted when this flag is set.
- NJD_EVAL_UNIT_ANG is set when the rotation amount is “0”. The rotation matrix calculation is omitted when this flag is set.
- NJD_EVAL_UNIT_SCL is set when the scale is “1” for x, y, and z. The scale matrix calculation is omitted when this flag is set.
- If NJD_EVAL_UNIT_POS, NJD_EVAL_UNIT_ANG, and NJD_EVAL_UNIT_SCL are all set, all matrix calculations are omitted, and the matrix push-pop operations are also omitted.
- The NJD_EVAL_HIDE flag is set by the user. If this flag is set, the model is not drawn. This flag is used when switching the gun or blade with which a model is equipped.
- The NJD_EVAL_BREAK flag is set by the user. If this flag is set, the child search is halted at this point. For example, setting this flag in the root node causes the entire model to disappear.
- When NJD_EVAL_BREAK is used in combination with motion, data coordination is lost; therefore, this flag should only be used in the root node. It can be used in intermediate nodes, but the user is responsible for such usage.
- The rotation evaluation sequence for LightWave3D is “ZXY”. Because this sequence is normally “ZXY” in Ninja, the NJD_EVAL_ZXY_ANG flag is provided for execution via a library with the LightWave3D evaluation sequence. When this flag is set to “ON”, the rotation calculation sequence is changed to “ZXY”.

Point structure

```
typedef struct {
Float      x;          /* X value */
Float      y;          /* Y value */
Float      z;          /* Z value */
} NJS_POINT3, NJS_VECTOR;
```

- Gives the X, Y, and Z values of a vertex.

Texture name structure

```
typedef struct {
void      *filename;      /* Texture file name */
Uint32    attr;          /* Texture attributes */
void      *texaddr;      /* Texture memory address */
} NJS_TEXNAME;
```

- Textures are specified by file name.
- “globalIndex” is a unique texture number specified by a Uint32-type variable. However, 0xffffffff through 0xffffffff cannot be used since the library uses them as internal flags.
- “globalIndex” is stored in the texture file. The “globalIndex” chunk is always placed at the start of a Ninja texture file.
- “globalIndex” is assigned and managed by this tool. In Ninja, this number is used to detect identical textures, thus avoiding duplicate registrations in texture memory.
- “attr” is used in the texture type and cache specifications.

```
#define NJD_TEXATTR_TYPE_FILE      0      /* File texture */
#define NJD_TEXATTR_CASHE        BIT_31
/* Registers texture in cache */
#define NJD_TEXATTR_TYPE_MEMORY BIT_30 /* Memory texture */
#define NJD_TEXATTR_BOTH        BIT_29
/* Registers texture in cache and texture memory */
#define NJD_TEXATTR_MASK 0xE0000000
```

- In a memory-type texture, it is necessary to set the texture color type and category code in “attr”. This is the same bit string that is set in the “.pvr” file texture type.

```
/* Color type */
#define NJD_TEXFMT_ARGB_1555      (0x00)
#define NJD_TEXFMT_RGB_565       (0x01)
#define NJD_TEXFMT_ARGB_4444     (0x02)
#define NJD_TEXFMT_YUV_422       (0x03)
#define NJD_TEXFMT_BUMP          (0x04)
#define NJD_TEXFMT_RGB_555       (0x05)
#define NJD_TEXFMT_COLOR_MASK    (0xFF)
/* Category code */
#define NJD_TEXFMT_TWIDDLED      (0x0100)
#define NJD_TEXFMT_TWIDDLED_MM  (0x0200)
#define NJD_TEXFMT_VQ           (0x0300)
#define NJD_TEXFMT_VQ_MM        (0x0400)
#define NJD_TEXFMT_PALETTIZE4    (0x0500)
#define NJD_TEXFMT_PALETTIZE4_MM (0x0600)
#define NJD_TEXFMT_PALETTIZE8    (0x0700)
#define NJD_TEXFMT_PALETTIZE8_MM (0x0800)
#define NJD_TEXFMT_RECTANGLE     (0x0900)
#define NJD_TEXFMT_STRIDE        (0x0B00)
#define NJD_TEXFMT_TYPE_MASK     (0xFF00)
```

- “texaddr” stores the texture memory address that is assigned when “texlist” is set in the current “texlist” in the target. This address is used in the current texture specification within the library.

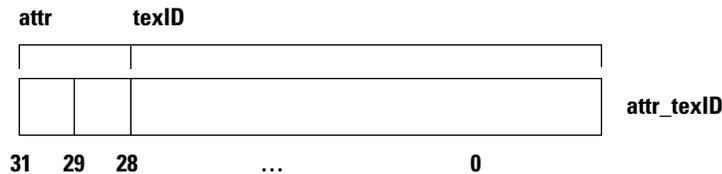
Texture list structure

```
typedef struct {
NJS_TEXNAME *textures; /* Texture name list */
Uint16      nbTexture; /* Number of textures */
} NJS_TEXLIST;
```

- This list is used to batch write multiple textures to texture memory. The library texture specification is made for each “texlist”.

Material structure

```
typedef struct {
NJS_COLOR   diffuse; /* Diffuse reflection (model color) 0 to 255 */
NJS_COLOR   specular; /* Specular reflection (highlights) 0 to 255 */
FLOAT       exponent; /* Highlight spread 0 to 300 */
Uint32      attr_texId; /* Attribute and texture ID */
Uint32      attrflags; /* Attribute flag */
} NJS_MATERIAL;
```



```
#define NJD_POLYGON_CALLBACK      (BIT_31)
#define NJD_MATERIAL_CALLBACK    (BIT_30)
#define NJD_CALLBACK_MASK       (BIT_31|BIT_30)
```

- “attr_texId” specifies a texture number in the current texture list “texlist”.
- The most significant portion of “attr_texId” specifies the use of the polygon callback routine and the material callback routine.
- The only texture information in the model tree that corresponds to a “texlist” entry number is “texId”. The user sets the “texlist” corresponding to the current model as the current texture list. For details on the attributes that are set in “attrflags”, refer to section 2.3, “Ninja Attributes.”

Meshset structure

```
typedef struct {
Uint16      type_matId; /* Type and material ID (0 to 4095) */
Uint16      nbMesh; /* Number of polygons/contiguous polygons */
Sint16      *meshes; /* Polygon list */
Uint32      *attrs; /* Polygon attributes */
NJS_VECTOR *normals; /* Polygon normal line vector list */
NJS_COLOR *vertcolor; /* Polygon vertex color list */
NJS_COLOR *vertuv; /* Polygon vertex UV list */
} NJS_MESHSET;
```

- The attributes of individual polygons are set in “attrs”. The attributes that are set in “attrs” are the same as those that are set in “attrflags” for “NJS_MATERIAL”.
- For details on meshsets, refer to the section, “Meshsets,” under the header, “Construction of a Model.”
- For details on the attributes that are set in “attrs”, refer to section, “Ninja Attributes.”

Model structures

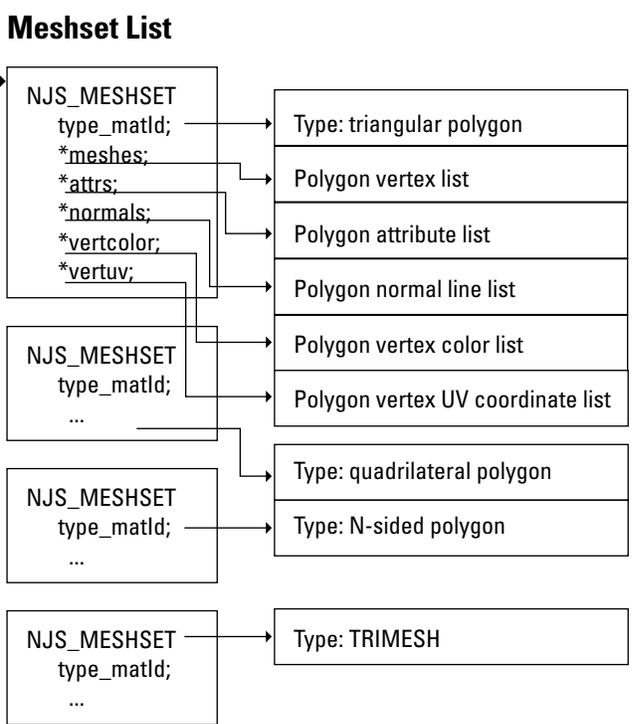
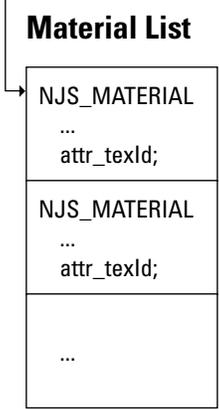
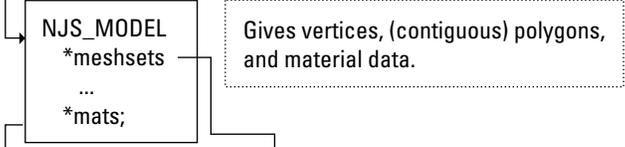
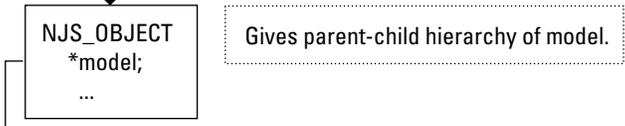
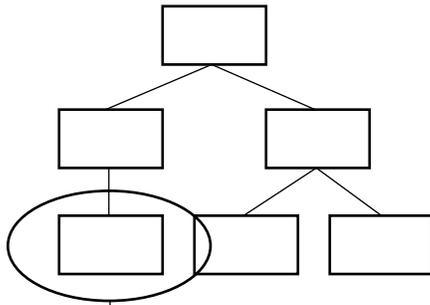
```
typedef struct {
NJS_POINT3      *points; /* Vertex list */
NJS_VECTOR      *normals; /* Vertex normal line vector list */
Uint32          nbPoint; /* Number of vertices */
NJS_MESHSET *meshsets; /* Polygon and TRIMESH list */
NJS_MATERIAL     *mats; /* Material list */
Uint16          nbMeshset; /* Number of meshsets; maximum: 65,535 */
Uint16          nbMat; /* Number of mats; maximum: 65,535 */
NJS_POINT3      center; /* Model center */
Float           r; /* Radius of circumscribed sphere from model center */
} NJS_MODEL;
```

- The vertex list includes all of the vertices used in multiple meshsets that are set in the MODEL structure.
- If vertex normal lines are not needed, set NULL in “normals”.
- “meshset” is a combined list of a single type of polygon (triangular polygons, quadrilateral polygons, N-sided polygons, TRIMESHes) that uses a single material.
- Each meshset has a material ID, and its position in the “mats” array can be specified.
- “center” and “r” are used when calculating model collisions, etc.

Construction of a Model

Diagram of Structure

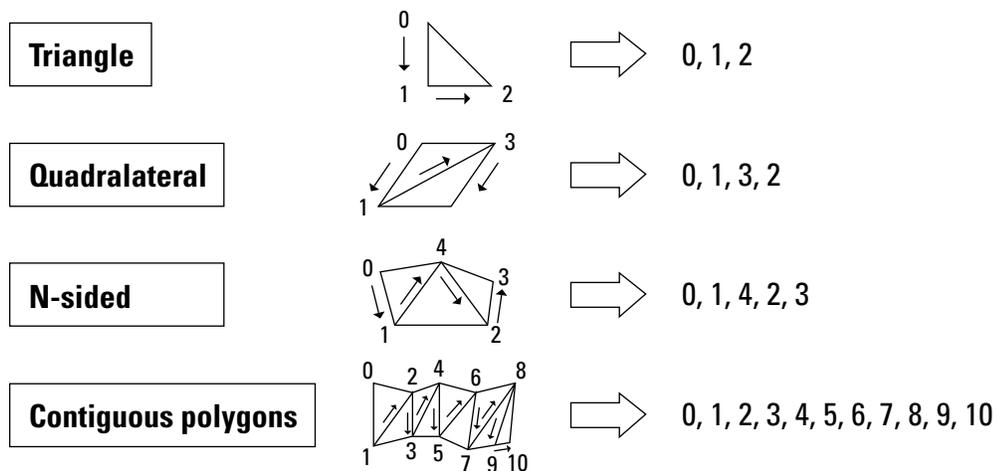
Object Tree



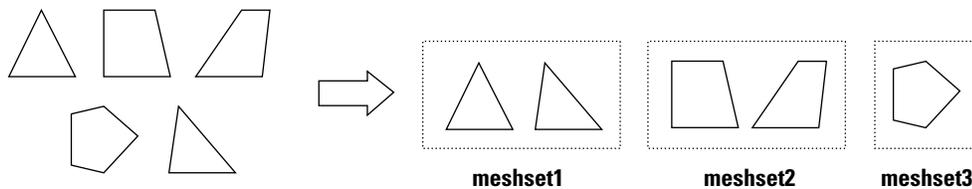
Meshsets

```
typedef struct {
  Uint16   type_matId; /* Type and material ID (0 to 16384) */
  Uint16   nbMesh; /* Number of polygons/contiguous polygons */
  Sint16   *meshes; /* Polygon list */
  Uint32   *attrs; /* Polygon attributes */
  NJS_VECTOR *normals; /* Polygon normal line vector list */
  NJS_COLOR *vertcolor; /* Polygon vertex color list */
  NJS_COLOR *vertuv; /* Polygon vertex UV list */
} NJS_MESHSET;
```

- This structure stores data strings for triangular polygons only, quadrilateral polygons only, N-sided polygons only, or TRIMESHes (contiguous polygons) only.
- The vertex sequence is the drawing (zig-zag) sequence for all of the contiguous polygons.



- Multiple types of meshset arrays are set for “*meshsets”.
- When the data includes triangular polygons, quadrilateral polygons, and N-sided polygons, the Ninja converter divides the data into separate meshsets according to the number of vertices.



- If multiple materials are used for triangular polygons (for example), the data is divided into separate meshsets for each material.
- In “type_matId”, the two most significant bits (bits 14 and 15) indicate the meshset type, while the 14 least significant bits (bits 0 to 13) indicate which material in the model structure material list is being used.

Example:

	Polygon1	Polygon2	
meshes[] =	{3, 4, 5, 9,	8, 6, 2, 10,	7, 13, 14, 11,}
attrs[] =	{0xffffffff, 0xffffffff, ...}		
vertcolor[] =	{0xFFFF, 0xEEEE, 0xCCCC, ...}		
vertuv[] =	{0xEFAB, 0xFF98, 0x44FF, ...}		
nbMesh =	number of vertices in "meshes" / 4		

- One attribute is allocated to each polygon. The attribute for the nth polygon is "attrs[n]". (n = 0, 1, 2, ...)
- One normal line is allocated to each polygon. The normal line for the nth polygon is "normals[n]". (n = 0, 1, 2, ...)
- The color and UV for the meshes[i] vertex are "vertcolor[i]" and "vertuv[i]", respectively. (i = 0, 1, 2, ...)
- The NULL pointer is set for "attrs", "normals", "vertcolor", and "vertuv" if they are not needed.

For a contiguous polygon list (NJD_MESHSET_TRIMESH)

- A continuous polygon is expressed by writing the number of vertices composing it at the beginning.

Example:

	trimesh1	trimesh2	
meshes[] =	{6, 3, 4, 5, 9, 8, 6, 4,	2, 10, 7, 13, 11}
attrs[] =	{0xffffffff, 0xffffffff, ...}		
normals[] =	{{1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, ...}		
vertuv[] =	{0xEFAB, 0xFF98, 0x44FF, ...}		
vertcolor[] =	{0xFFFF, 0xEEEE, 0xCCCC, ...}		
nbMesh =	Number of trimeshes		

- One polygon attribute is allocated to each trimesh. The attribute for the nth trimesh is "attrs[n]". (n = 0, 1, ...)
- Although the normal line of a trimesh is usually derived from the external product, the normal line can be stored as data in "normals".
- One normal line is allocated to each polygon after conversion to triangular polygons. The normal line for the nth triangular polygon is "normals[n]". (n = 0, 1, 2, ...)
- The color and UV of the vertex of meshes[i] are respectively vertcolor[i-(k+1)] and vertuv[i-(k+1)] (i=0, 1, 2, ...). Here, k is the current trimesh number (the kith trimesh).
- The NULL pointer is set for "attrs", "normals", "vertcolor", and "vertuv" if they are not needed.

For an N-sided polygon list (NJD_MESHSET_N)

- Here, “N” represents a value of “5” or more. In other words, this declaration is used to generate a polygon with five or more sides. In the future, it will be possible to generate lists of polygons with three or more sides through a converter option.
- It is important to note that in the case of an N-sided polygon, the “meshes” vertex number will deviate from the “vertcolor” and “vertuv” numbers

Example:

```
          Polygon1      Polygon2
meshes[ ] = { 5, 3, 4, 5, 9, 8, 7, 6, 2, 10, 7, 13, 14, 11, ... }
```

- The underlined value indicates the number of vertices (N), and is followed by N vertices.

```
attrs[ ] = {0xffffffff, 0xffffffff, ...}
```

- One polygon attribute is allocated to each polygon. The attribute for the nth polygon is “attrs[n]”. (n = 0, 1, 2, ...)

```
normals[ ] = {{1.0,0.0,0.0}, {0.0, 1.0, 0.0}, ...}
```

- A normal line vector is assigned to each polygon. The normal line for the kth polygon is “normals[k]”. (k = 0, 1, 2, ...)
- Each vertex of an N-sided polygon is assumed to lie on the same plane, so the normal line that is derived from the first three points is regarded to be the normal line for the entire polygon.

```
vertcolor[ ] = {0xFFFF,0xEEEE,0xCCCC, ...}
```

```
vertuv[ ] = {0xEFAB,0xFF98,0x44FF, ...}
```

- The color and UV for the meshes[i] vertex are “vertcolor[i-(k+1)]” and “vertuv[i-(k+1)]”, respectively. (k = 0, 1, 2, ...; i = 0, 1, 2, ...) Here, “k” is the number of the current (“kth”) polygon. This is because “meshes” has the value “N” that indicates the number of sides for each polygon, while “vertcolor” and “vertuv” do not.

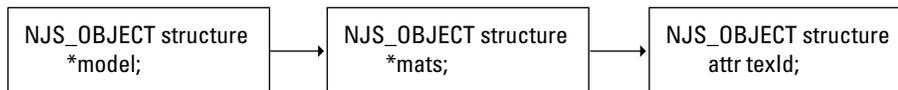
```
nbMesh = Number of N-sided polygons
```

- The NULL pointer is set for “attrs”, “normals”, “vertcolor”, and “vertuv” if they are not needed.

Construction of a Texture

Diagram of Structure

Object Tree



This indicates the number of the texture to be used from "texlist".
This structure has no "texlist" data; it uses the "texlist" that is set as the current "texlist".

Texlist



```

typedef struct {
    NJS_TEXNAME *textures; /* Texture name list */
    Uint16      nbTex;     /* Number of textures */
} NJS_TEXTLIST;
  
```



```

typedef struct {
    void *filename; /* Texture file name */
    Uint32 attr; /* Texture attributes */
    void *texaddr; /* Texture attributes */
} NJS_TEXNAME;
  
```

texlist

- Textures are managed through real files and texlists.
- Texture memory is overwritten in units of whole texlists.
- Memory data can be specified instead of a file.

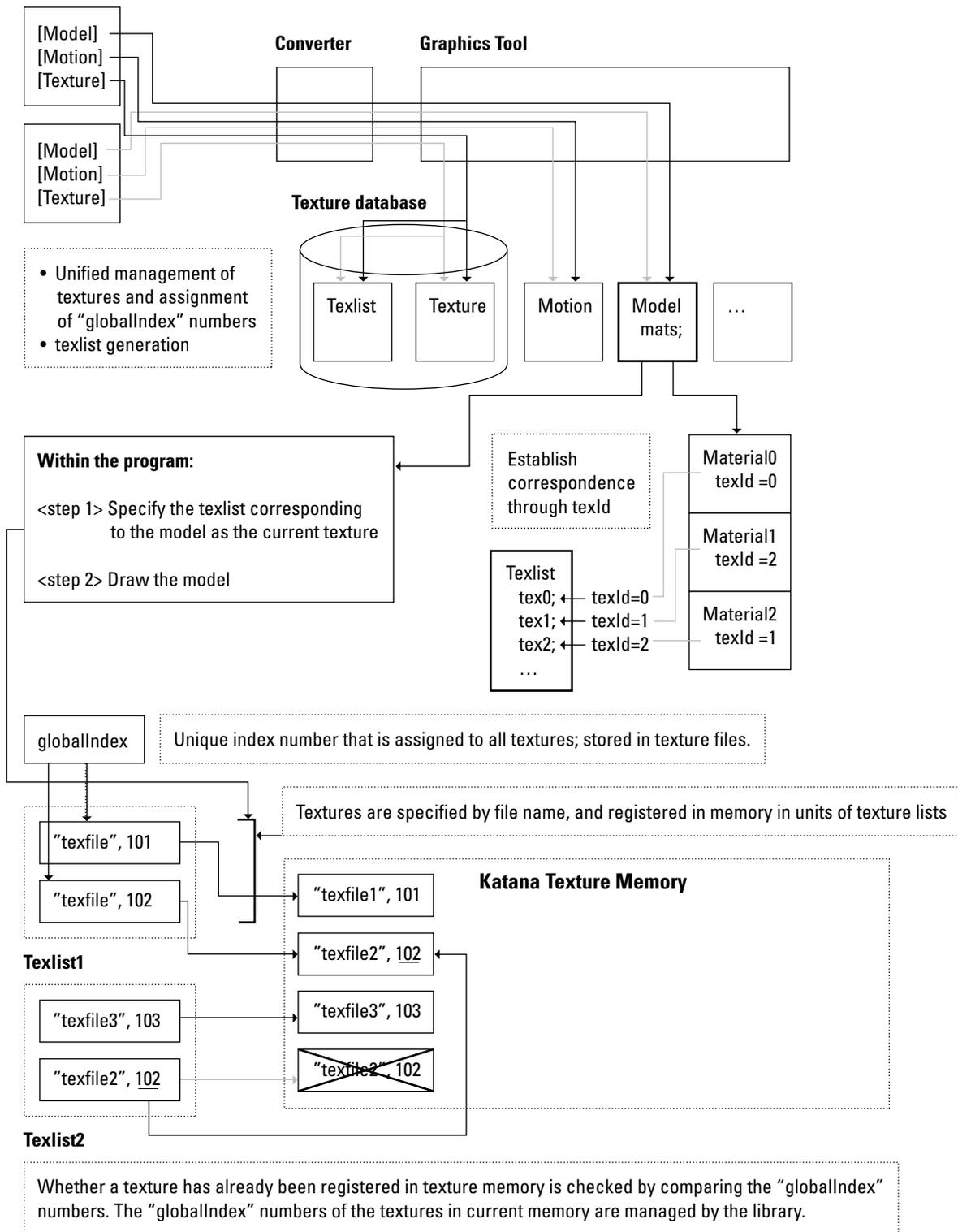
globalIndex

- "globalIndex" is a unique number that is assigned to all textures.
- Texture duplication is checked by comparing the "globalIndex" numbers.
- "globalIndex" is stored in a texture file.

texaddr

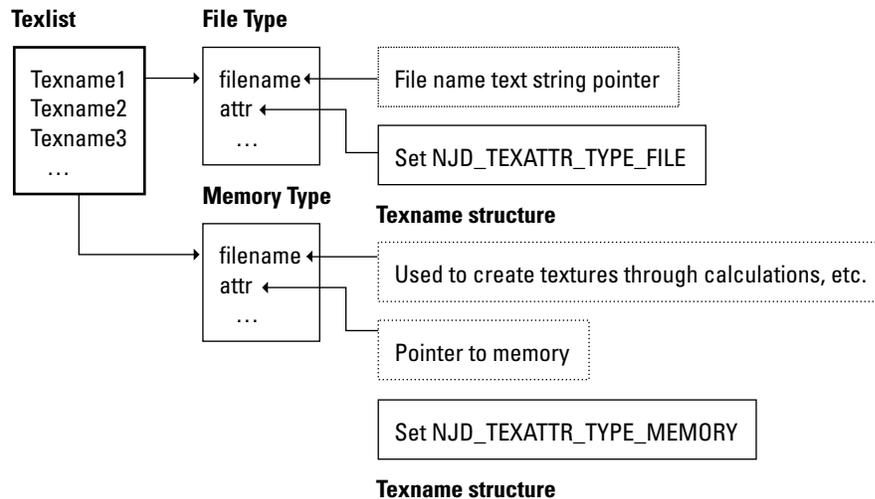
- The current texture is set in terms of individual texlists by the library function. The registered address is then stored in texture memory.

Overview of Texture Processing



Memory-type Textures and the Texture Cache

For details, refer to the library specifications. The following is an overview only.



When the NJD_TEXATTR_CASHE flag is set in "attr", only the texture cache is set. If NJD_TEXATTR_BOTH is set, then when textures are registered in texlist units in texture memory, any textures in the cache are automatically registered in texture memory from the cache.

Explanation of the structure of textures

- Normally, multiple textures are applied to a model; the texlist structure is defined in order to handle these models as a batch.
- The texlist structure consists of an array of multiple texture file names.
- The "globalIndex" numbers are stored in texture files, and are retrieved when the file is loaded.
- The "globalIndex" chunk is located at the top of the texture file. The "globalIndex" numbers are required, because they are essential for improved library performance.
- The "globalIndex" numbers are managed by the Ninja graphics tool, and are assigned in such a manner that there are no duplicates within an entire project.
- When multiple texlists that contain the same texture are loaded into memory, duplicates are detected through their "globalIndex" numbers.
- During model conversion, all of the texture files in the texture group used in the object tree are output as a single texlist.
- When model conversion is repeated, if the textures that appear had "globalIndex" numbers assigned to them previously, those same "globalIndex" numbers are assigned to them again.
- If the user assigns his own "globalIndex" numbers, he creates a table of all of the textures that are used and then writes those entry numbers in the "globalIndex" chunks of all of the texture files.
- The model texture information bears no direct relationship with the "globalIndex" numbers since the information is expressed by texIds only, which are just the texlist entry numbers.
- The correspondence between textures and models is established before the model is drawn by setting as the current texture the texlist that is to be used. Textures can then be easily substituted by changing the texlist...
- During current texture registration, the address in texture memory is stored in texaddr. This address is used by the library.

- Because the texture file is assumed to reside in a specific folder, the path name for a texture file is not included in the texture file name description in texlist.
- The texture extension “.pvr” is omitted in order to reduce the amount of data.
- The user is responsible for maintaining agreement between texlists and models (object trees). In order to improve performance, the library does not detect disagreements between the number of textures in a texlist and the number of textures used in an object tree.

Ninja Attributes

The attributes that are defined here are used in both “attrflags” for “NJS_MATERIAL” and “attrs” in “NJS_MESHSET”.

If the attributes “attrs” for an individual polygon is NULL, the “attrflags” setting for the material is valid.

<u>31-29</u>	<u>28-26</u>	25	24	23	22	21	<u>20</u>	19	<u>18-17</u>	<u>16-15</u>	14-13	<u>12</u>	<u>11-8</u>	7	6-0
--------------	--------------	----	----	----	----	----	-----------	----	--------------	--------------	-------	-----------	-------------	---	-----

<u>31-29</u>	: SRC Alpha Instruction	(Alpha blending parameter; explained later)
<u>28-26</u>	: DST Alpha Instruction	(Alpha blending parameter; explained later)
25	: Ignore Lights	(Light source enabled / disabled; disabled when “1”)
24	: Flat Shading	(Flat shading ON / OFF)
23	: Double Side	(Double side polygon ON / OFF)
22	: Environment Mapping	(Environment mapping ON / OFF)
21	: Use Texture	(Texture enabled / disabled; enabled when “1”)
<u>20</u>	: Use Alpha	(Alpha enabled / disabled; enabled when “1”)
19	: Ignore Specular	(Ignores specular; disabled when “1”)
<u>18-17</u>	: Flip UV	(Flip control)
<u>16-15</u>	: Clamp UV	(Clamp control)
14-13	: Filter-Mode	0 ... Point Sampled(hard spec) 1 ... Bilinear Filter(hard spec) 2 ... Tri-liner Filter(hard spec) 3 ... Blend Texture(Ninja flag) Explained later
<u>12</u>	: Super-Sample Texture	(Anisotropic Filter ON / OFF)
<u>11-8</u>	: Mip-Map eDi adjust	(16-step mip-map adjustment; explained later)
7	: Pick Status	(Stores the status that has been picked)
6-0	: User Flags	(Available to user; used in combination with callback)

- If “blend texture” is set for “Filter-mode”, the current polygon is drawn twice using the material that follows the current “matId” (in other words, the “matID + 1” material).
- The default user mask is 0x000007f. The user can specify this value through the library. This value is specified through the following library functions (tentative names):

NjSetPolygonAttrUserMask(usermask)

This sets the user mask for “attrs” in NJS_MESHSET. This value affects individual polygons.

```
NjSetMaterialAttrUserMask(usermask)
```

This sets the user mask for “attrflags” in NJS_MATERIAL. This value affects individual materials.

For example, if mip-map “D” adjust (11 to 8) is not to be controlled for individual polygons, setting “0x00000fff” as the user mask allows the user to control bits 6 through 0 as well as another 5 bits (11 through 7), for a total of 12bits, for his own purposes. These flags are evaluated by the user through the callback routine for individual polygons. The library sets flags that retained globally in sections other than “usermask”. Because “attr” is also specified for individual materials, if “usermask” is set the value that is set in “material” is reflected in the settings for each polygon. In the same fashion, the user can also evaluate user flags in the material callback routine. Although “usermask” can be changed dynamically (for individual model trees, for example), the user must be responsible for their control. It is possible to evaluate the material “attrflags” user bits in the material callback routine and make changes to the polygon “usermask”.

- Alpha blending parameter

In the blending function, two RGBA values and SRC and DST, are combined as described below, and the result is returned to DST.

```
DST := SRC * BlendFunction(SRC Alpha Instruction) +
      DST * BlendFunction(DST Alpha Instruction)
```

Here, a three-bit instruction is input along with the SRC and DST colors in BlendFunction (Instruction). This function then returns coefficients that have been weighted by the four alpha values for each RGBA.

Instruction	Field Value	Values Returned
Zero	0	(0, 0, 0, 0)
One	1	(1, 1, 1, 1)
‘Other’ Color	2	(O _R , O _G , O _B , O _A)
Inverse ‘Other’ Color	3	(1 - O _R , 1 - O _G , 1 - O _B , 1 - O _A)
SRC Alpha	4	(S _A , S _A , S _A , S _A)
Inverse SRC Alpha	5	(1 - S _A , 1 - S _A , 1 - S _A , 1 - S _A)
DST Alpha	6	(D _A , D _A , D _A , D _A)
Inverse DST Alpha	7	(1 - D _A , 1 - D _A , 1 - D _A , 1 - D _A)

“Other Color” and “Inverse Other Color” indicate that the DST color is to be used if specified in the SRC instruction, or that the SRC color is to be used if specified in the DST instruction.

The addition operation is performed after the coefficients have been determined and the SRC/DST multiplication operations have been performed. In this case, overflow checking and clamping of the result that was obtained are performed when appropriate.

- Filter Mode

Field Values	Filter Mode
0	Point Sampled
1	Bilinear Filter
2	Tri-linear
3	Texture Blend

“Texture blend” specifies a combination of the “matId” and “matId + 1” textures while the bi-linear filter is on. This function can be used for bump mapping and texture blending. This function alters only the current structure. This function will be implemented in the current and subsequent versions.

- Mip-Map “D” adjust

Although the mip-map “D” value is calculated by the drawing engine internally, there are instances where fine adjustments are made forcibly in order to find meeting points between aliasing and blurring. These adjustments are made by multiplying the computed “D” value by the specified adjustment value (a 4-bit unsigned fixed-decimal value with a 2-bit decimal field).

Example ‘D’ Adjust bit pattern	Equivalent value
00.00	<i>Illegal*</i>
00.01	0.25
01.00	1.0
11.11	3.75

* Specification not permitted

The Ninja flags are defined below. They are labelled as flags, except for those fields that consist of two or more bits. Other portions are defined separately. Although the UV Flip and Clamp fields are two-bit fields, they are regarded as flags for U and V, separately. Numerous masks used for extracting these flags are also defined.

```
/* SRC Alpha Instr(31-29) */
#define NJD_SA_ZERO      (BIT_0)                /* 0 zero                */
#define NJD_SA_ONE      (BIT_29)               /* 1 one                 */
#define NJD_SA_OTHER    (BIT_30)               /* 2 Other Color         */
#define NJD_SA_INV_OTHER (BIT_30|BIT_29)       /* 3 Inverse Other Color */
#define NJD_SA_SRC      (BIT_31)               /* 4 SRC Alpha           */
#define NJD_SA_INV_SRC  (BIT_31|BIT_29)       /* 5 Inverse SRC Alpha   */
#define NJD_SA_DST      (BIT_31|BIT_30)       /* 6 DST Alpha           */
#define NJD_SA_INV_DST  (BIT_31|BIT_30|BIT_29) /* 7 Inverse DST Alpha   */
#define NJD_SA_MSAK     (BIT_31|BIT_30|BIT_29) /* MASK                  */

/* DST Alpha Instr(31-29) */
#define NJD_SD_ZERO      (0)                   /* 0 zero                */
#define NJD_SD_ONE      (BIT_26)               /* 1 one                 */
#define NJD_SD_OTHER    (BIT_27)               /* 2 Other Color         */
#define NJD_SD_INV_OTHER (BIT_27|BIT_26)       /* 3 Inverse Other Color */
#define NJD_SD_SRC      (BIT_28)               /* 4 SRC Alpha           */
#define NJD_SD_INV_SRC  (BIT_28|BIT_26)       /* 5 Inverse SRC Alpha   */
#define NJD_SD_DST      (BIT_28|BIT_27)       /* 6 DST Alpha           */
#define NJD_SD_INV_DST  (BIT_28|BIT_27|BIT_26) /* 7 Inverse DST Alpha   */
#define NJD_SD_MSAK     (BIT_28|BIT_27|BIT_26) /* MASK                  */

/* filter mode */
#define NJD_FILTER_POINT (0)
#define NJD_FILTER_BILINEAR (BIT_13)
#define NJD_FILTER_TRILINEAR (BIT_14)
#define NJD_FILTER_BLEND (BIT_14|BIT_13)
#define NJD_FILTER_MASK (BIT_14|BIT_13)

/* Mip-Map ëDí adjust */
#define NJD_D_025      (BIT_8)                /* 0.25                  */
#define NJD_D_050      (BIT_9)                /* 0.50                  */
#define NJD_D_075      (BIT_9|BIT_8)          /* 0.75                  */
#define NJD_D_100      (BIT_10)               /* 1.00                  */
#define NJD_D_125      (BIT_10|BIT_8)         /* 1.25                  */
#define NJD_D_150      (BIT_10|BIT_9)         /* 1.50                  */
#define NJD_D_175      (BIT_10|BIT_9|BIT_8)   /* 1.75                  */
#define NJD_D_200      (BIT_11)               /* 2.00                  */
#define NJD_D_225      (BIT_11|BIT_8)         /* 2.25                  */
#define NJD_D_250      (BIT_11|BIT_9)         /* 2.50                  */
#define NJD_D_275      (BIT_11|BIT_9|BIT_8)   /* 2.75                  */
#define NJD_D_300      (BIT_11|BIT_10)        /* 3.00                  */
#define NJD_D_325      (BIT_11|BIT_10|BIT_8)  /* 3.25                  */
#define NJD_D_350      (BIT_11|BIT_10|BIT_9)  /* 3.50                  */
#define NJD_D_375      (BIT_11|BIT_10|BIT_9|BIT_8) /* 3.75                  */
```

```
#define NJD_D_MASK      (BIT_11|BIT_10|BIT_9|BIT_8)      /* MASK */

/* flags */
#define NJD_FLAG_IGNORE_LIGHT      (BIT_25)
#define NJD_FLAG_USE_FLAT          (BIT_24)
#define NJD_FLAG_DOUBLE_SIDE      (BIT_23)
#define NJD_FLAG_USE_ENV          (BIT_22)
#define NJD_FLAG_USE_TEXTURE      (BIT_21)
#define NJD_FLAG_USE_ALPHA        (BIT_20)
#define NJD_FLAG_IGNORE_SPECULAR  (BIT_19)
#define NJD_FLAG_FLIP_U           (BIT_18)
#define NJD_FLAG_FLIP_V           (BIT_17)
#define NJD_FLAG_CLAMP_U          (BIT_16)
#define NJD_FLAG_CLAMP_V          (BIT_15)
#define NJD_FLAG_USE_ANISOTROPIC  (BIT_12)

/* Flip and clamp masks */
#define NJD_FLAG_FLIP_MASK (NJD_FLAG_FLIP_U| NJD_FLAG_FLIP_V)
#define NJD_FLAG_CLAMP_MASK \
    (NJD_FLAG_CLAMP_U| NJD_FLAG_CLAMP_V)
/* Mask for flags that are sent directly to the hardware */
#define NJD_FLAG_HARD_MASK (NJD_FLAG_USE_ALPHA \
    | NJD_FLAG_FLIP_MASK | NJD_FLAG_CLAMP_MASK \
    | NJD_FLAG_USE_ANISOTROPIC)
/* Mask for flags that are evaluated by the library (i.e., masks that are not sent directly
to the hardware) */
#define NJD_FLAG_SOFT_MASK ( NJD_FLAG_IGNORE_LIGHT \
    | NJD_FLAG_USE_FLAT| NJD_FLAG_DOUBLE_SIDE \
    | NJD_FLAG_USE_ENV| NJD_FLAG_USE_TEXTURE \
    | NJD_FLAG_IGNORE_SPECULAR|NJD_FLAG_PICK)
/* Mask for all flags */
#define NJD_FLAG_MASK (NJD_FLAG_HARD_MASK \
    | NJD_FLAG_SOFT_MASK)
/* Default user mask */
#define NJD_DEFAULT_USER_MASK \
    (BIT_6|BIT_5|BIT_4|BIT_3|BIT_2|BIT_1|BIT_0)
/* Default system mask */
#define NJD_DEFAULT_SYS_MASK \
    ~NJD_DEFAULT_USER_MASK
/* Mask for fields that are sent as is to the hardware */
#define NJD_SYS_HARD_MASK (NJD_SA_MASK|NJD_SD_MASK \
    |NJD_FLAG_HARD_MASK|NJD_D_MASK)
```

Texture Format

The “.pvr” format is used. The converter is “pvrconv”. Textures that are embedded in the model converter and are automatically used in models are wholly converted.

The converter checks the alpha value of the original image, switches the format automatically to one of the following three formats, and then outputs the image.

- If there is no alpha value: Outputs the image in RGB565 format.
- If there is an alpha value: Outputs the image in ARGB4444 format.
- If the alpha value is 0 or 255: Outputs the image in ARGB1555 format.

In addition, if the texture is square, the converter automatically selects twiddled format; if the texture is rectangular, the converter automatically selects rectangle format.

<twiddled format>

With this texture, the pixels are arranged in the order in which they were read out of memory at high speed. Mip-map can be used. Display is fast.

<rectangle format>

With this texture, the pixel order is that of the image. Display is slow, compared to twiddled format. Note that mip-map cannot be used.

<Bump mapping>

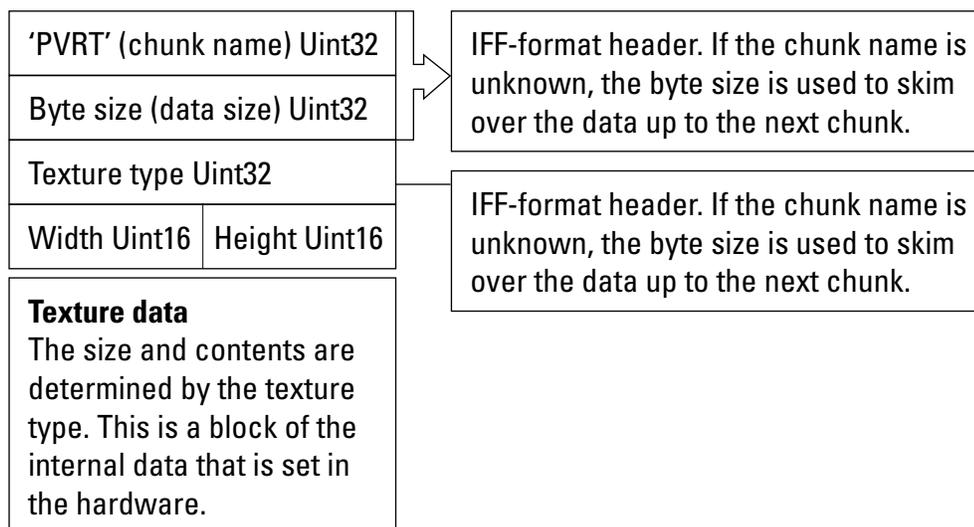
The bump mapping texture is provided for gray scale images. This texture cannot handle RGB color images. The converter converts the data to a format that is expected by the hardware.

The specifications for YUV422, palettes, and the VQ texture are currently under study.

<Texture format>

An overview of the texture format follows. The format is an IFF-based chunk format (header + size + data). The data portion consists of the internal data structure as is, as expected by the hardware. Details of this format are not covered here.

The chunk format is as follows:



The bit string that is produced by ORing the color type with the category code is set for the texture type.

```
/* Color type */
#define NJD_TEXFMT_ARGB_1555          (0x00)
#define NJD_TEXFMT_RGB_565           (0x01)
#define NJD_TEXFMT_ARGB_4444        (0x02)
#define NJD_TEXFMT_YUV_422          (0x03)
#define NJD_TEXFMT_BUMP              (0x04)
#define NJD_TEXFMT_RGB_555           (0x05)
#define NJD_TEXFMT_COLOR_MASK       (0xFF)
/* Category code */
#define NJD_TEXFMT_TWIDDLED          (0x0100)
#define NJD_TEXFMT_TWIDDLED_MM      (0x0200)
#define NJD_TEXFMT_VQ                (0x0300)
#define NJD_TEXFMT_VQ_MM            (0x0400)
#define NJD_TEXFMT_PALETTIZE4        (0x0500)
#define NJD_TEXFMT_PALETTIZE4_MM    (0x0600)
#define NJD_TEXFMT_PALETTIZE8        (0x0700)
#define NJD_TEXFMT_PALETTIZE8_MM    (0x0800)
#define NJD_TEXFMT_RECTANGLE         (0x0900)
#define NJD_TEXFMT_STRIDE            (0x0B00)
#define NJD_TEXFMT_TYPE_MASK         (0xFF00)
```

Aside from the PVRT chunk, the GBIX and PVRI chunks are defined in Ninja.

'GBIX' (chunk name) Uint32
4 (byte) Uint32
globalIndex Uint32

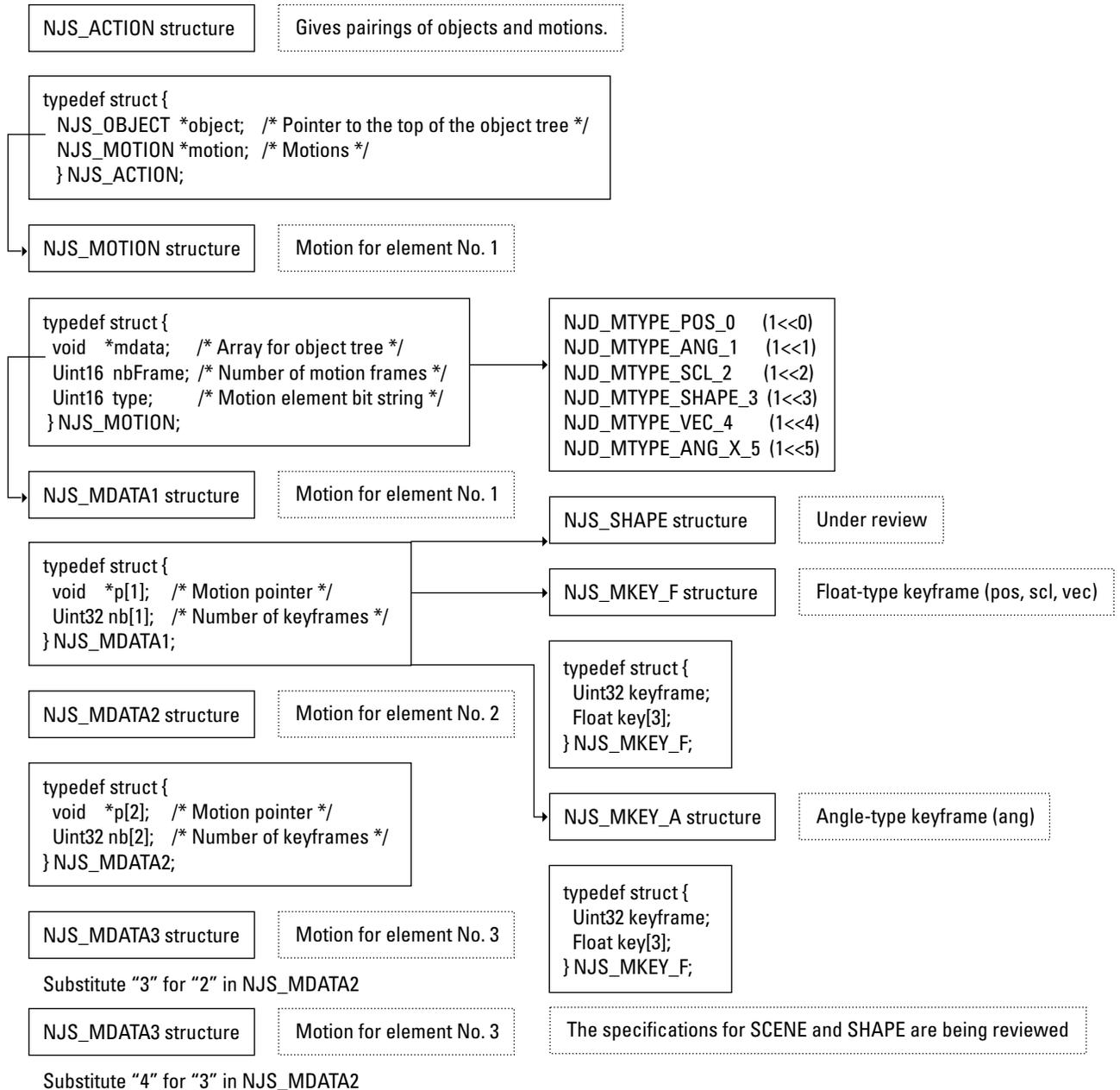
Specifies the “globalIndex” for the texture. When using a pvr file in Ninja, this chunk is placed at the top of the file.

'PVRI' (chunk name) Uint32
Byte size Uint32
Data
Texture control data

This filed stores the texture control data. The details are currently under study.

Motion Structures

Diagram of Structure



Explanation of Structures

Action structure

```
typedef struct {  
NJS_OBJECT *object; /* Pointer to the top of the object tree */  
    NJS_MOTION *motion; /* Motion list */  
} NJS_ACTION;
```

- “object” has a tree structure with a parent-child hierarchy.
- “motion” sets the motion that is to be applied to “object”.

Motion structure

```
typedef struct {  
void *mdata; /* Array for object tree */  
uint16 nbFrame; /* Number of motion frames */  
uint8 type; /* Motion element bit string */  
uint8 inp_fn; /* Interpolation method and number of elements */  
NJS_MOTION;
```

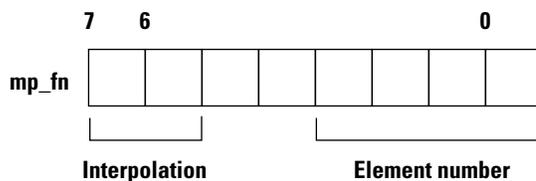
- “mdata” contains, in the form of an array, a number of NJS_MDATA sufficient for all of the NJS_OBJECTs included in the object tree.
- For NJS_MDATA, the NJS_MDATA1 to 4 structures are used according to the number of motion configuration elements.
- In addition, NJS_MDATA4 can always be used to set NULL for a pointer that has no data.
- At present, the following types will be supported: “pos”, “ang”, “scl”, “shape”, and “vec”.

```
#define NJD_MTYPE_POS_0 (1<<0) /* Use NJD_MKEY_F */  
#define NJD_MTYPE_ANG_1 (1<<1) /* Use NJD_MKEY_A */  
#define NJD_MTYPE_SCL_2 (1<<2) /* Use NJD_MKEY_F */  
#define NJD_MTYPE_SHAPE_3 (1<<3) /* Under review */  
#define MJD_MTYPE_VEC_4 (1<<4) /* Use NJD_MKEY_F */  
#define MJD_MTYPE_ANG_X_5 (1<<5) /* Use NJD_MKEY_AX */
```

- When normal motion includes the four elements of parallel motion (“pos”), rotation (“ang”), scale (“scl”) and SHAPE, NJS_MDATA4 is used. The number at the end of the label gives the order of the motion elements.
- The vector component “vec” is used in conjunction with “pos” in light source and camera motion.
- The interpolation calculation method is specified by the two most significant bits of “inp_fn”.

```
#define NJD_MTYPE_LINER 0x00 /* Linear interpolation */  
#define NJD_MTYPE_SPLINE 0x40 /* Spline interpolation */  
#define NJD_MTYPE_USER 0x80 /* User function interpolation */  
#define NJD_MTYPE_MASK 0xc0 /* Sampling mask */
```

- The element number that indicates which structure is being used is stored in the least significant four bits of “inp_fn”.



NJS_MDATA1 to 4 structures

```
typedef struct {  
void    *p[1]; /* Motion pointer */  
Uint32  nb[1]; /* Number of keyframes */  
} NJS_MDATA1;
```

```
typedef struct {  
void    *p[2]; /* Motion pointer */  
Uint32  nb[2]; /* Number of keyframes */  
} NJS_MDATA2;
```

```
typedef struct {  
void    *p[3]; /* Motion pointer */  
Uint32  nb[3]; /* Number of keyframes */  
} NJS_MDATA3;
```

```
typedef struct {  
void    *p[4]; /* Motion pointer */  
Uint32  nb[4]; /* Number of keyframes */  
} NJS_MDATA4;
```

- All data is expressed in the structure of a keyframe.
- “nb[i]” contains the number of motion keyframes for element p[i].

Key structures

```
typedef struct {  
Uint32  keyframe; /* Keyframe number */  
Float   key[3]; /* Float-type key value */  
} NJS_MKEY_F;
```

- Used in parallel motion (“pos”), scale (“scl”), and vector (“vec”).

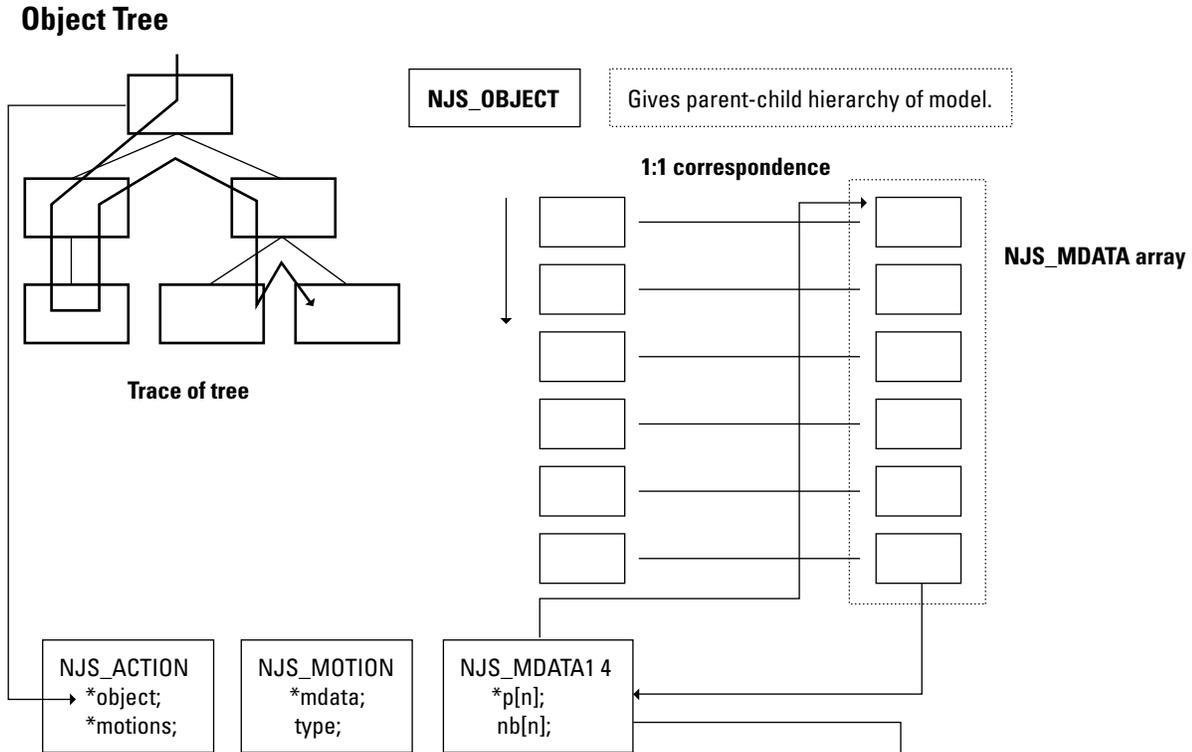
```
typedef struct {  
Uint32  keyframe; /* Keyframe number */  
Angle   key[3]; /* Angle-type key value */  
} NJS_MKEY_A;
```

- Used in rotation (“ang”).

```
typedef struct {  
Uint32  keyframe; /* Keyframe number */  
Angle   angle; /* Rotation angle */  
Float   key[3]; /* Rotation axis vector */  
} NJS_MKEY_AX;
```

Construction of a Motion

Diagram of Structure



Example: For "pos" only, NJS_MDATA1 is used because there is only one element.
 NJS_MOTION structure type = NJD_MKEY_POS_0;
 NJS_MKEY_F pos[] = { , , ...};
 NJS_MDATA1 mdata[] = {{pos, poskey_n}, ...};

Example: For "pos", "ang", and "scl", NJS_MDATA3 is used because there are three elements.
 type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1 | NJD_MTYPE_SCL_2;
 MKEY_F pos[] = { , , ...};
 MKEY_A ang[] = { , , ...};
 MKEY_F scl[] = { , , ...};
 MDATA3 mdata[] = {{pos, ang, scl, poskey_n, angkey_n, sclkey_n}, ...};

Example: For "pos" and "vec" for a light source, NJS_MDATA2 is used because there are two elements.
 type = NJD_MTYPE_POS_0 | NJD_MTYPE_VEC_4;
 NJS_MKEY_F pos[] = { , , ...};
 NJS_MKEY_F vec[] = { , , ...};
 NJS_MDATA2 mdata[] = {{pos, vec, poskey_n, veckey_n}, ...};

Explanation of Structure

- The specifications for SCENE and shape are currently under review.
- All motion is given by keyframe data.
- The user executes motion using linear interpolation of keyframe data and spline.
- The interpolation method can be defined by the user in the Ninja library through the callback function.
- The keyframe numbers start from zero.

position	Parallel motion
Angle	Rotation
Scale	Enlargement/reduction
shape	Animation by moving the vertices of a polygon. Specifications under review.
vector	Implements light source animation when used in combination with "position."

- The five motion elements are listed above.
- "vec" is used for light sources only, and "shape" is used for models only. Therefore, motion can have a maximum of four elements. These are defined in the NJS_MDATA1 to 4 structures.
- The pointers for the storage of each NJS_MDATA element are void, and in all cases it is necessary to stipulate the data storage order.

```
#define NJD_MTYPE_POS_0    (1<<0) /* Use NJS_MKEY_F */
#define NJD_MTYPE_ANG_1    (1<<1) /* Use NJS_MKEY_A */
#define NJD_MTYPE_SCL_2    (1<<2) /* Use NJS_MKEY_F */
#define NJD_MTYPE_SHAPE_3 (1<<3) /* Under review */
#define NJD_MTYPE_VEC_4    (1<<4) /* Use NJS_MKEY_F */
```

- The numbers indicated at the end of the "define" character string indicate the order of the data, with the newest data coming first. The above flags are set to the motion structure member type.

Example: For "pos" and "ang"

```
type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1;
mdata[] = {pos, ang, ...}
```

- The motion interpolation method is specified by the two most significant bits of "type".

```
#define NJD_MTYPE_LINER    0x0000
#define NJD_MTYPE_SPLINE  0x4000
#define NJD_MTYPE_USER     0x8000
#define NJD_MTYPE_MASK     0xc000
```

- NJD_MTYPE_LINER indicates linear interpolation.
- NJD_MTYPE_SPLINE indicates spline interpolation.
- NJD_MTYPE_USER indicates interpolation through a user-defined routine.
- The root is "pos" and "ang"; in other cases, such as an "ang"-only motion model, the NJS_MDATA2 structure is used. A non-root "pos" is handled by using the NULL pointer.

```
type = NJD_MTYPE_POS_0 | NJD_MTYPE_ANG_1;
NJS_MDATA2 mdata[] = {
    { *pos1, *ang1 },
    { NULL, *ang2 },
    { NULL, *ang3 },
    .... }

```

- Note that in the above example, “ang2” and “ang3” must not be directly adjacent to “NULL”.



05. Scroll Guide

Revision Information

Ver.0.04

The member clip of the scroll structure can not be used.

Ver.0.05

- The description of the member clip in "Scroll-related Structure" were changed.
- Color Definition" was modified.

Image Units as Related to Scrolling

Overview

This chapter explains the image units which Ninja uses in scrolling.

Image Units

Pixel

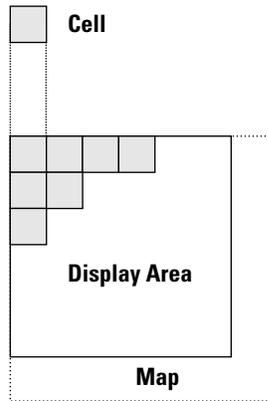
The smallest component unit of an image

Cell

The smallest unit of an image which makes up a scrolling screen. Cells in Ninja are composed of between 8 and 1024 pixels. The maximum number of Cells which a program can hold is defined by NJD_CELL_NUM_MAX.

Map

Maps are composed of collections of Cells. The maximum number of maps which a program can hold is defined by `NJD_MAP_MAX`.



Scroll Rotation, Resizing, and Movement

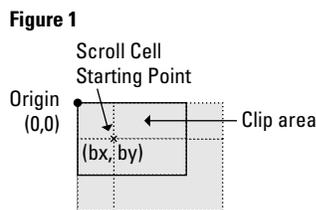
Overview

This chapter shows the meanings of the various values used in setting scroll displays and scroll structures, and how those values are calculated.

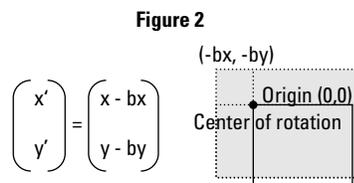
Scroll Rotation, Resizing, and Movement

Scroll Rotation, Resizing, and Movement are described as follows.

- 1) Both the scroll area and clip area use the upper left corner of the screen as the origin. The x and y coordinates which mark the starting point of a scroll cell are designated bx, by (see Figure 1).

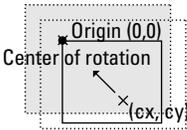


- 2) Points on the Scroll Display move in bx, by from the origin by $-bx, -by$ (see Figure 2).



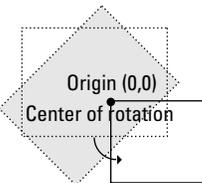
- 3) To perform rotations, move toward The origin by the difference of the center of rotation (cx, cy) (see Figure 3).

Figure 3

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix}$$


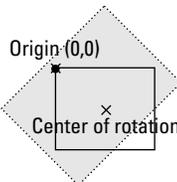
- 4) Make the center of rotation into the origin, and rotate via the matrix m (see Figure 4).

Figure 4

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = m \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix}$$


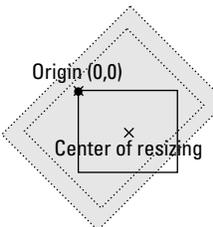
- 5) After rotation, restore to original by degree that Scroll Display was moved (see Figure 5).

Figure 5

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = m \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix}$$


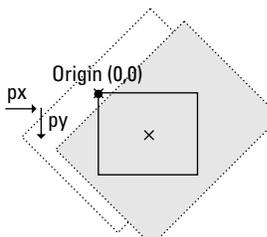
- 6) Resize by sx, sy, centering on the center of resizing (spx, spy) (see Figure 6).

Figure 6

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = m \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix} - \begin{pmatrix} spx \\ spy \end{pmatrix} \begin{pmatrix} sx \\ sy \end{pmatrix} + \begin{pmatrix} spx \\ spy \end{pmatrix}$$


- 7) Finally, move by px, py (see Figure 7).

Figure 7

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = m \begin{pmatrix} x - bx - cx \\ y - by - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix} - \begin{pmatrix} spx \\ spy \end{pmatrix} \begin{pmatrix} sx \\ sy \end{pmatrix} + \begin{pmatrix} spx \\ spy \end{pmatrix} + \begin{pmatrix} px \\ py \end{pmatrix}$$


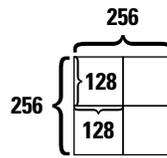
Scroll Programming

Overview

In this chapter, we cover everything from drawing of Cells to depicting Scroll area.

Example of Programming a Scroll

- 1) Draw a Cell Image- Draw a Cell image by following the texture creation rules. Note that Cell size must be from 8 to 1024. Ex.: Draw four 128x128 textures in one 256x256 texture



- 2) Convert Cell Image to pvr format- Use tools to convert textures to PVR format.
- 3) Create Texture List- Create the texture name structure and texture list structure. Refer to "Texture Guide" in Chapter 2 of this book for the details on how to create them.
- 4) Creating a Map- Create the following map as an example.

0	1	4	5	16
2	3	6	7	17
8	9	12	13	18
10	11	14	15	19

The map data comes from the previously created files in the following order.

0, 1, 2, 3 from test0.pvr

4, 5, 6, 7 from test1. pvr

8, 9, 10, 11 from test2. pvr

12, 13, 14, 15 from test3. pvr

16, 17, 18, 19 from test4. pvr

Texture numbers are taken in the order they were stored in the list's creation, starting with test0.pvr. test0.pvr is 0, and test4.pvr is 4. We use this and the mapmaking macro NJM_MAP to create maps. NJM_MAP is NJM_MAP(texture number, texture U, texture V). Thus, the 0 area of the map is NJM_MAP(0, 0, 0), 1 is NJM_MAP(0, 128, 0), etc. The map array that results from this is...

```

Uint32 map[4][5] ={
{NJM_MAP(0,0,0), NJM_MAP(0,128,0), NJM_MAP(1,0,0), NJM_MAP(1,128,0), NJM_MAP(4,0,0)},
{NJM_MAP(0,0,128), NJM_MAP(0,128,128),NJM_MAP(1,0,128), NJM_MAP(1,128,128), NJM_MAP(4,128,0)},
{NJM_MAP(2,0,0), NJM_MAP(2,128,0), NJM_MAP(3,0,0), NJM_MAP(3,128,0), NJM_MAP(4,0,128)},
{NJM_MAP(2,0,128), NJM_MAP(2,128,128),NJM_MAP(3,0,128),NJM_MAP(3,128,128), NJM_MAP(4,128,128)}
};

```

5) Define the Scroll Structure

Define all the elements of the Scroll Structure

celps	assigns cell pixel size between 8 and 1024.
mapw	assigns map width in number of Cells
maph	assigns map height in number of Cells
sw	assigns horizontal scroll display image size.
sh	assigns vertical scroll display image size
list	assigns pointer to texture list structure
map	assigns pointer to top address of map array. Make sure that map is at least of dimensions map[maph][mapw]. Anything smaller will leave this variable undefined
px,py	assigns coordinates for movement of scroll display
bx,by	assigns coordinates for beginning of map draw
pr	assigns scroll priority
sflag	sets resize flag (ON, OFF).
sx,sy	sets the ratio for x- and y-axis resizing
spx,spy	assigns coordinates for center of resizing area
mflag	sets rotation matrix flag (ON, OFF).
cx,cy	assigns coordinates for center of rotation area
m	assigns rotation matrix.
colmode	assigns color mode
colmix	assigns color computations (unimplemented at present)
clip[2]	Not used in this version
attr	attribute (unimplemented at present)
sclc	applies color to entire scroll. Varies according to color mode

6) Use Scroll Functions

Finally, we will try out the scroll functions (using the map and texture list previously created).

First, load the textures.

```
njInitTexture(&texmemlist,5);  
njLoadTexture(&texlist);
```

Assign scroll structure (see "Scroll function, Structures, and Definitions")

```
scl.celyps = 128;  
:  
:(omitted)
```

Using the scroll functions, you can draw the scrolls

```
njDrawScroll (&scl);
```

Color

Overview

This chapter explains about color modes which can be used in colmode of scroll structures

Color Mode

NJD_COLOR_MODE_FLAT_TEXTURE

This mode is used when "No translucent" (RGB565) is set for the textures of all cells.

NJD_COLOR_MODE_FLAT_TEXTURE_TRANS

This mode is used when some (even if only one) of textures of the cell is "translucent" (ARGB1555 or ARGB 4444) .

Scroll function, Structures, and Definitions

Overview

This chapter explains Ninja scroll functions, scroll structures, and scroll definitions.

Scroll-related Functions

njDrawScroll

Draws 2D scroll

Format

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

Parameters

*scl scroll structure pointer

Return value

none

Function

Draws 2D scroll in clip display

Notes

For details on creating textures, refer to the Texture document.

Scroll-related Structure

```
NJS_SCROLLStructure
typedef struct {
    Uint16      celps;      /* Cell Pixel size */
    Uint16      mapw,maph; /* Number of Cells */
    Uint16      sw,sh;     /* Scroll display image size */
    NJS_TEXLIST list;     /* Pointer to texture list structure */
    Uint16      *map;      /* Pointer to top address of map array */
    Float       px,py;     /* Coordinates for drawing scroll */
    Float       bx,by     /* Coordinates for drawing scroll origin */
    Float       pr;       /* Priority */
    Sint16      sflag;    /* Resize flag (ON, OFF) */
    Float       sx,sy;    /* x- and y-axis resizing ratio */
    Float       spx,spy;  /* center of resizing area */
    Sint16      mflag;    /* Rotation flag (ON, OFF) */
    Float       cx,cy;    /* Center of rotation area */
    NJS_SCLMTRX m;       /* Rotation Matrix */
    Uint16      colmode;  /* Color Mode */
    Uint16      colmix;   /* Color Computations (unimplemented at present) */
    NJS_POINT2  clip[2]  /* Clip point */
    NJS_SCLATTR attr;    /* Attribute */
    NJS_COLOR   sclc;    /* ITE Color */
}NJS_SCROLL;
```

Scroll-related Definitions

Maximum Values

```
#define NJD_CELL_NUM_MAX    0xFFFF /* the maximum of cell's number */
#define NJD_MAP_W_MAX      0xFF    /* the maximum of map's width */
#define NJD_MAP_H_MAX      0xFF    /* the maximum of map's height */
#define NJD_MAP_MAX        (NJD_MAP_W_MAX*NJD_MAP_H_MAX)
```

Color definitions (colormode)

```
#define NJD_COLOR_MODE_PACKED_TEXTURE    33
#define NJD_COLOR_MODE_PACKED_TEXTURE_TRANS    41
```

Texture Structures for Use in Cell Programming

NJS_TEXINFOstructure

```
typedef struct{
    void*                texaddr;    /* texture memory address cache */
    NJS_TEXSURFACE      texsurface;
} NJS_TEXINFO;
```

NJS_TEXNAMEstructure

```
typedef struct{
    void    *filename;    /* Pointer to filename or NJS_TEXINFO structure */
    Uint32  attr;        /* Texture Attributes */
    Uint32  texaddr;     /* Texture Address */
}NJS_TEXNAME;
```

NJS_TEXLISTstructure

```
typedef struct {
    NJS_TEXNAME    *textures;    /* texture array */
    Uint32         nbTexture;    /* texture count */
} NJS_TEXLIST;
```




06. Nindows Tutorial

1. Nindows Tutorial

1.1 Summary

Nindows is an easy to use GUI system for performing tasks essential to game development such as debugging and adjusting parameters on the actual machine and the host machine.

1.2 Special Features of Nindows

- You can use the same controls as those in Windows and other common GUIs.
- Windows can be freely created in an application with the Nindows API.
- Handy utilities can be used in debugging the Texture Viewer and other areas without complicated programming.
- Parameters adjusted with Nindows can be confirmed in real time, allowing rapid adjustment of game balance.
- Adjusted parameters can be saved to a file on the host machine or to backup memory on the actual machine. (Not supported in this version)

2 Creating a Simple Nindows Application

This chapter explains how to integrate Nindows into an existing Ninja application. Nindows functions can be easily enabled by adding just a few line changes to the source file.

2.1 Integrating Nindows

Preparing the Ninja application.

Get the source file which contains the functions `njUserInit()`, `njUserMain()`, and `njUserExit()`.

Include the Nindows header file

Add the following line to the source file.

```
#include <Nindows.h>
```

Call the Nindows initialization function

After the call to `njInitTexture()`, add the following line.

```
nwInitSystem(numTextures);
```

`numTextures` is the number of texture memory lists.

It assigns the value specified in `njInitTexture()`.

Call the function to execute Nindows

Change the last instance of return `NJD_USER_CONTINUE` in `njUserMain` to the following line.

```
return nwExecute();
```

Call the function to exit Nindows

Before the call to `njExitSystem()` add the following line.

```
nwExitSystem();
```

Linking the Nindows Library

- Add `Nindows.lib` to the project.

The preceding steps enable:

- 1) Use of Nindows' Nindows Utility
- 2) Calls to Nindows API functions

The result of the preceding steps is the following source code.

```
#include <Nindows.h>

NJS_TEXMEMLIST tex[1000];

void njUserInit(void)
{
    njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555,1);
    njInitVertexBuffer(500000, 0, 500000, 0);
    nwInitSystem(1000);
}

Sint32 njUserMain(void)
{
    return nwExecute();
}

void njUserExit(void)
{
    nwExitSystem();
}
```

2.2 Description of Functions used in Integrating Nindows

Function	Description
NwInitSystem	Initializes the Nindows system
NwExitSystem	Exits the Nindows system
NwExecute	Draws all windows
NwInitResource	Loads textures used in Nindows

nwInitSystem	Initialization function
Format	void nwInitSystem(UINT32 numTextures)
Parameters	NumTextures Number of texture memory lists
Return value	None
Function	<ul style="list-style-type: none"> • Initializes the Nindows system and enables Nindows utilities and Nindows API functions. • Please assign the same value to numTextures as was assigned in njInitTexture().
Reference	NwExecute(),nwExitSystem(),nwInitResource(),njInitTexture(),njClipZ()

nwInitSystem	Initialization function
Note	<ul style="list-style-type: none">• Automatically loads the textures used in Nindows.• When using the njReleaseAllTexture() Ninja function, call nwInitResource() and reload the textures.• Nindows reserves texture global index numbers 0xffffffff0 to 0xfffffffffe, so applications cannot use textures stored in this range.
Example	<pre>#define MAX_TEXTURE 1000 static NJS_TEXMEMLIST texlist[MAX_TEXTURE]; void njUserInit(void) { njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1); njInitVertexBuffer(500000, 0, 500000, 0); njInitTexture(texlist, MAX_TEXTURE); nwInitSystem(MAX_TEXTURE); }</pre>

NwExitSystem	Initialization function
Format	void nwExitSystem(void)
Parameters	None
Return value	None
Function	Exits Nindows.
Reference	njExitSystem()
Note	
Example	<pre>void njUserExit(void) { nwExitSystem(); nwExitSystem(); }</pre>

nwExecute	Execution Function
Format	Sint32 nwExecute(void)
Parameters	None
Return value	If 'Exit' is selected from the System Menu it returns NJD_USER_EXIT, in all other cases it returns NJD_USER_CONTINUE.
Function	Performs all Nindows drawing.
Reference	njUserMain()
Note	<ul style="list-style-type: none">• Call once each frame.• When this function is called, the Nindows system draws all windows.• As in the example below, when the function's return value is used as the return value for njUserMain(), the application can be exited by selecting the 'Exit' menu.
Example	<pre>Sint32 njUserMain(void) { : : : return nwExecute(); }</pre>

nwInitResource	Initialization Function
Format	Void nwInitResource(void)
Parameters	None
Return value	None
Function	Loads the textures used in Nindows
Reference	nwInitSystem(),njInitTexture(),njReleaseTexture()
Note	<ul style="list-style-type: none">• Usually there is no need to use this function, but when njReleaseTextureAll() is used as in the example, the textures used by Nindows are also released. Therefore, you should always call this function after calling njReleaseTextureAll().
Example	<pre>njReleaseTextureAll(); nwInitResource();</pre>

3. Using Nindows and Nindows Utilities

3.1 Using Nindows

In the last section, the integration of Nindows was completed. When a Nindows integrated application (hereafter Nindows application) is executed, the mouse cursor is displayed on the screen and moves on the screen in response to mouse manipulation.

System Menu

When the right mouse button is clicked on the desktop, a popup menu is displayed. This is called the System Menu from which menus can be selected to use Nindows utilities. The System Menu contains the following items.

Menu Item	Description
Debug	Displays the Nindows Utility menu.
User(Undefined)	Displays a user-defined menu. At the time of Nindows initialization, the User Menu is not entered in the system, so it is displayed in a light color and cannot be selected.
Font	Change Font
Exit	Exits the application.

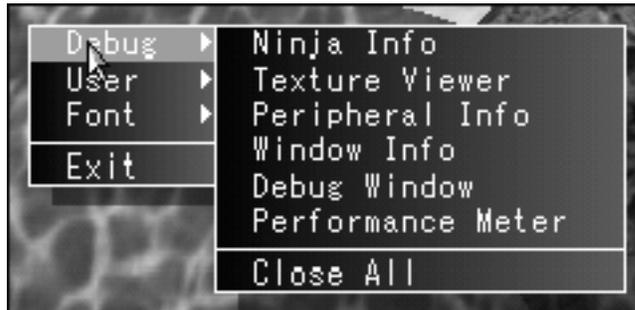


Diagram 3-1. The System Menu is displayed with the right button and the "Debug" menu is selected

3.2 Nindows Utilities

The menu items that are displayed when "Debug" is selected from the System Menu are the Nindows utilities. Nindows contains the following utilities.

Name	Description
Ninja Info	Displays the Ninja library version number, and other information.
Texture Viewer	All of the textures which are read in can be displayed.
Peripheral Info	Displays information about peripherals.
Window Info	Displays information about the active windows
Debug Window	A handy window for displaying debug messages.
Performance Meter	Describes the application's drawing performance.

Ninja Info Window



Diagram 3-2. Ninja Info Window

The following information is displayed in the Ninja Info Window.

Display	Contents
Ninja Ver.	Ninja library version number
Nindows Ver.	Nindows version number
Vertex	Number of vertices
Calc polygon	Number of polygons
Draw polygon	Number of draw polygons
Texture Memory	Amount of Texture Memory available and total memory

Texture Viewer Window

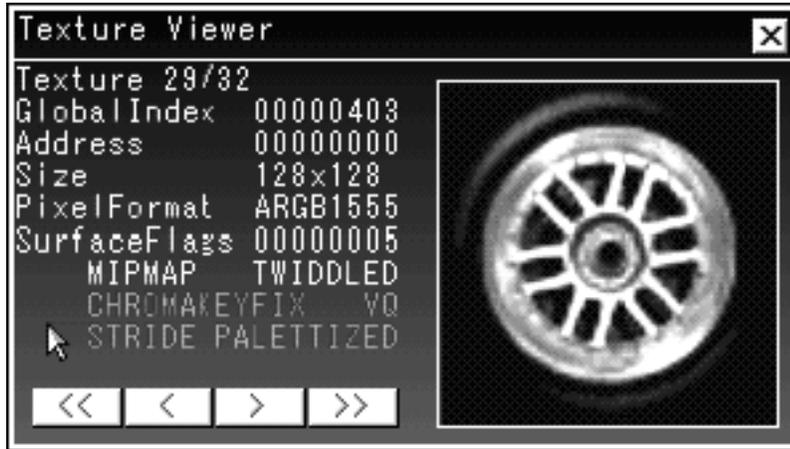


Diagram 3-3. Texture Viewer Window

- All entered textures can be viewed.
- The textures can be changed via four buttons. The following information is displayed in the Texture Viewer Window.

Display	Contents
Texture	Texture numbers and the total number of textures.
GlobalIndex	Global Index
Address	Texture addresses
Size	Texture size
PixelFormat	Pixel format
SurfaceFlags	Surface flag: The highlighted items are the flags for the texture. Refer to the texture related document for the details.
Memory Flag	Similarly, this flag is set by the texture. Refer to the texture documentation for details on this flag.
Error Code	This error code is for texture loading. "OK" is displayed when loading succeeds. Refer to the texture documentation for error code details.

Peripheral Info Window



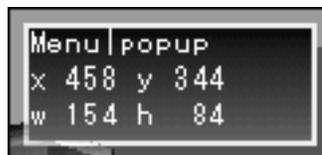
Diagram 3-4. Peripheral Info Window

This window displays information about peripherals (input devices). Peripheral ports can be selected using the [<] and [>] buttons.

The Peripheral Info Window displays the following information

Display	Contents
Port	Peripheral port name
Dev	Name of the peripheral attached to the port
ON	Info about the button being pressed
OFF	Info about the button not being pressed
PRESS	Info about the button the moment it is pressed
RELEASE	Info about the button the moment it is released
X	X axis value
Y	Y axis value

Window Info Window



Peripheral Info Window

- This window displays information about the window under the mouse cursor (the active window).
- It can also be used for debugging things like application windows created with the Nindows API.
- The Peripheral Info Window displays the following information.

Display	Contents
	Title of the active window
x,y	Upper left coordinates of the window's client area
w,h	Size of the window's client area

*The x, y coordinates are not the absolute screen coordinates, it depends on the window style.

Debugging Window

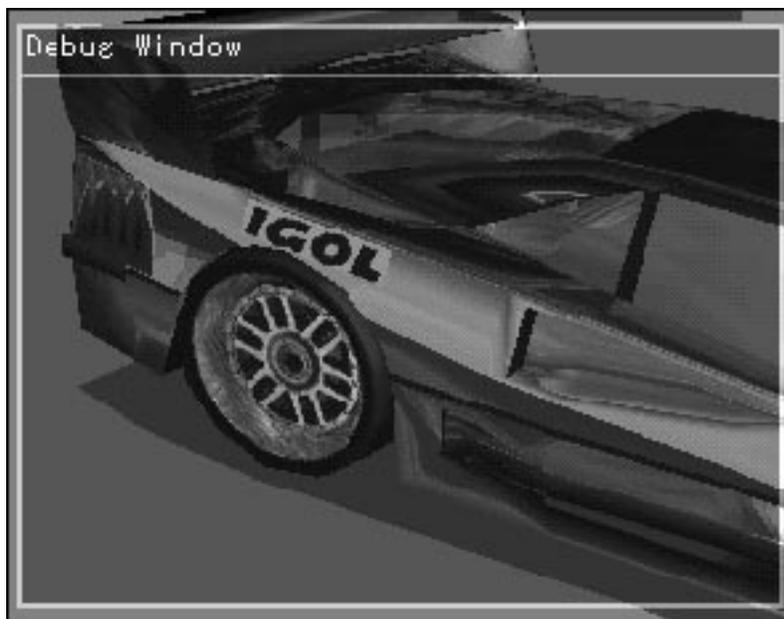
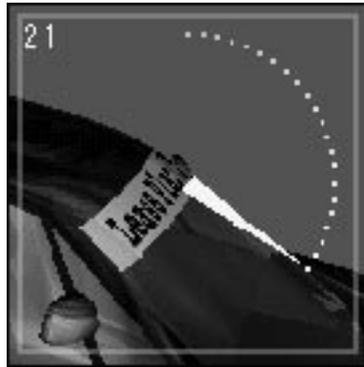


Diagram 3-5. Debugging Window

- At first, nothing is displayed in this window.
- Display the debugging characters using the `nwDebugPrintf()` function.
- This function can be used in the same way as the standard `printf()` function.
- For more details, please refer to the Edit Window chapter.

Performance Meter Window



Performance Meter Window

- Provides an intuitive understanding of an application's performance (calculation, drawing speed).
- If the meter revolves once per second, the frame rate is 60fps.

Changing Fonts

Select 'Font' from the System menu to select normal or large font size. If normal size is hard to read on an NTSC monitor, select a larger font size.

* This version does not automatically resize the window according to changes in font size. If you change the font, do so before opening another window after executing the application.

4. Windows

4.1 Summary

The window is the most fundamental element of Nindows. An application draws to the window's client area by creating a window and specifying a drawing callback function. Or it can create controls such as another window, a button, or scrollbar and control it as a child window.

4.1.1 Types of Windows and Window Classes

The following are the types of windows divided into Window Classes.

Window Class	Window Type
NWD_WC_WIN	Standard window
NWD_WC_SCRWIN	Window with scrolling enabled in the client area
NWD_WC_EDITWIN	Edit window
NWD_WC_SCROLLBAR	Scrollbar control
NWD_WC_BUTTON	Button control
NWD_WC_MENUWIN	Menu window

The next section will mainly discuss the Standard window.

4.2 Creating a Window

For example, let's create a window on the desktop which displays a counter in the client area. To create the window, we will use the function `nwCreateWindow()`.

```
static void test_window_callback(NHWND hWnd)
{
    static int count = 0;
    char buf[256];
    sprintf(buf, "Count = %d", count++);
    nwTextOut(hWnd, 1, 1, buf);
}

Bool create_test_window(void)
{
    NHWND hWnd = nwCreateWindow(NWD_WC_WIN,
                               "Test Window",
                               NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                               50, 50, 100, 100,
                               NULL);

    if (!hWnd) return FALSE;
}
```

```
        hWnd->clientDraw = test_window_callback;
        return TRUE;
    }
    void njUserInit(void)
    {
        :
```

Now, a window will be displayed on the screen at the specified location, and the counter display will be incremented. Furthermore, the window will be displayed in the next frame after calling `nwExecute()` and the return from `njUserMain()`.

The window is destroyed by calling the function `nwDestroyWindow()` or by clicking on the close box in the caption bar with the mouse. (Only windows which have `NWD_WS_CONTROL` specified in their window style have a close box.)

```
nwDestroyWindow(hWnd);
```

4.3 Creating a Child Window

The last argument in `nwCreateWindow()` is a window handle for a parent window. In the last example, this argument was set to `NULL`, so we created a window which did not have a parent (actually, the Desktop Window is a parent). The next example shows how to create a parent window and a child window.

```
static void test_window_callback(NWHWND hWnd)
{
    static int count = 0;
    char buf[256];
    sprintf(buf, "Count = %d", count++);
    nwTextOut(hWnd, 1, 1, buf);
}

Bool create_test_window(void)
{
    NWHWND hWndParent, hWnd;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                               "Parent Window",
                               NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                               50, 50, 100, 100,
                               NULL);
    if (!hWndParent) return FALSE;

    hWndParent->clientDraw = test_window_callback;

    //The code below is for child window creation
    hWnd = nwCreateWindow(NWD_WC_WIN,
                          "Child Window",
                          NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_OFFSET,
                          20, 20, 60, 60,
                          hWndParent);
    if (!hWnd) return FALSE;

    return TRUE;
}
```

When a parent window is destroyed using the `nwDestroyWindow()` function or by mouse operation, all of its child windows are automatically destroyed. In this example, if the parent window `hWndParent` is destroyed, the child window `hWnd` is also destroyed.

```
static void test_window_callback(NHWND hWnd)
```

Furthermore, scrollbars (class `NWD_WC_SCRBAR`), buttons (class `NWD_WC_BUTTON`) and menus (class `NWD_WC_MENUWIN`) cannot be specified as a parent window (they cannot have a child window). However, it is possible for a menu to have a child menu (sub menu). For more details, please refer to chapter 9.

4.4 Window Related Parameters

The window handle `NHWND` is actually a pointer to the `NWS_WIN` structure.

By directly setting this structure's members through the handle of the window created, the window can be made to do various actions. Here we will discuss members which are useful to know and representative ways of using them.

> Client drawing callback function `hWnd->clientDraw`

- Used in the window creation example, it is the most representative member.
- Usually, some kind of callback function address is set in this member, and the drawing to the client area is processed within that callback function.

> Destructor `hWnd->destructor`

- If a function address is set in this member, it will be called back when the window is destroyed.

> User Data `hWnd->param1, hWnd->param2`

- A `Sint32` type member which can be freely set and referenced in the application.

> User Data `hWnd->userBuf`

- When you want to save a lot of user data, this data address is specified.
- Please reserve a separate data buffer in the application.

4.5 Description of Window Support Functions

Function	Description
<code>nwCreateWindow</code>	Creates a window
<code>nwDestroyWindow</code>	Destroys a window

Nindows API

nwCreateWindow Window Creation Functions

Format NHWND nwCreateWindow(Sint32 wClass, Sint8* caption, Sint32 style, Sint32 x, Sint32 y, Sint32 w, Sint32 h, NHWND hWndParent)

Parameters
wClass- Window class
caption- Window name (caption)
style- Window style
x,y- Upper left coordinate of the client area
w,h- hWndParent Parent window handle

Return value If successful, it returns the handle of the window created. If the window could not be created, NULL is returned.

Function Creates a window

Reference nwDestroyWindow(),nwCreateMenuWindow(),nwCreateEditWindow(),nwCreateScrollBar(),nwCreateButton() NWS_WIN structure

Note

- For creating menu windows, edit windows, scrollbars, and buttons it is recommended to use the more convenient nwCreateMenuWindow(),nwCreateEditWindow(),nwCreateScrollBarArray(),nwCreateButton().

Example

```
// Creates a window
NHWND hWnd;
hWnd = nwCreateWindow(NWD_WC_WIN,
    "Test Window",
    NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
    50, 50, 100, 100,
    NULL);
```

The following flags are set in the Window style.

Window Style	Meaning
NWD_WS_CAPTION	Has a caption
NWD_WS_BORDER	Has a thin border line
NWD_WS_THICKFRAME	Has a thick, resizable border line
NWD_WS_SHADING	Window color can be set at each vertex
NWD_WS_CONTROL	Has a close box
NWD_WS_SHADOW	Window has a shadow
NWD_WS_INVISIBLE	Creates an invisible window
NWD_WS_NOMOVE	Cannot be moved with the mouse
NWD_WS_OFFSET	Creates a window in a position (x, y) relative to the parent window

If `NWD_WC_SCROLLBAR` is specified in the Window class, please also specify one of the following flags.

Window Style	Meaning
<code>NWD_WS_SB_HORZ</code>	Create a horizontal scrollbar
<code>NWD_WS_SB_VERT</code>	Create a vertical scrollbar

<code>nwDestroyWindow</code>	Window Creation Function
Format	<code>void nwDestroyWindow(NWHWND hWnd)</code>
Parameters	<code>hWnd</code> - the handle of the window to be destroyed
Return value	None
Function	Destroys the window
Reference	<code>NwCreateWindow()</code> , <code>NWS_WIN</code> structure
Note	<ul style="list-style-type: none">• If a callback function is set in <code>hWnd->desructor</code>, it calls back that function.
Example	<pre>// Destroys a window nwDestroyWindow(hWnd);</pre>

Callback Functions

<code>ClientDrawCallback</code>	Window Callback Function
Format	<code>Void ClientDrawCallback(NWHWND hWnd)</code>
Parameters	<code>HWnd</code> - Handle of the window where the callback originated
Return value	None
Function	An application defined function which a window calls back for drawing
Reference	<code>nwCreateWindow()</code> , <code>NWS_WIN</code> structure
Note	

DestroyCallback	Window Callback Function
Format	void DestroyCallback(NHWND hWnd)
Parameters	hWnd- Handle of the window where the callback originated
Return value	None
Function	Application defined function called back when a window is destroyed
Reference	nwDestroyWindow(), NWS_WIN structure
Note	

4.4 Samples and a Description of Window Support Functions

In addition to nwCreateWindow() and nwDestroyWindow() the Nindows API has many functions to support the management of windows. The sample below uses a joystick to move a window.

```
Sint32 njUserMain(void)
{
    Sint32 x, y, w, h, style;
    NHWND hWnd;
    NJS_PERIPHERAL* per = njGetPeripheral(NJD_PORT_JOYSTICK1);

    // Checks to see if the window exists
    if (!(hWnd = nwFindWindow(NULL, "Test Window"))) goto skip;

    // Acquires the coordinates and size of the window's client area
    nwGetWindowPos(hWnd, &x, &y);
    nwGetWindowSize(hWnd, &w, &h);
    if (per->press & NJD_DGT_ST) {
        //When the start button is pressed, the window is moved to the center of the screen
        nwSetWindowPos(hWnd, 320 - w / 2, 240 - h / 2);
    }
    if (per->on & NJD_DGT_TA) {
        // If the A button is pressed, the window can be moved with the cross key
        if (per->on & NJD_DGT_KU) y--;
        if (per->on & NJD_DGT_KD) y++;
        if (per->on & NJD_DGT_KL) x--;
        if (per->on & NJD_DGT_KR) x++;
        nwSetWindowPos(hWnd, x, y);
    }
    if (per->on & NJD_DGT_TB) {
        // If the B button is pressed, the window's size can be changed with the cross key
        if (per->on & NJD_DGT_KU) h--;
        if (per->on & NJD_DGT_KD) h++;
        if (per->on & NJD_DGT_KL) w--;
        if (per->on & NJD_DGT_KR) w++;
    }
}
```

```
        nwSetWindowSize(hWnd, w, h);
    }
    if (per->press & NJD_DGT_TC) {
        // When the C button is pressed, the window caption bar can be switched ON/OFF
        nwGetWindowStyle(hWnd, &style);
        if (style & NWD_WS_CAPTION) {
            nwSetWindowStyle(hWnd, ~NWD_WS_CAPTION, 0);
        } else {
            nwSetWindowStyle(hWnd, 0xffffffff, NWD_WS_CAPTION);
        }
    }
}

skip:
    return nwExecute
```

Function	Description
nwFindWindow	Searches the window with the specified caption
nwFindWindowByPos	Searches the window in the specified location
nwGetClientRect	Gets the rectangle of the specified window's client area
nwGetWindowColor	Gets the color of the specified window
nwGetWindowPos	Gets the upper left coordinates of the specified window's client area
nwGetWindowRect	Gets the overall rectangle of the specified window
nwGetWindowSize	Gets the width and height of the specified window's client area
nwGetWindowStyle	Gets the style of the specified window
nwGetWindowText	Gets the caption string of the specified window
nwSetWindowColor	Changes the color of the specified window
nwSetWindowPos	Sets the upper left coordinates of the client area and moved the specified window
nwSetWindowSize	Changes the width and height of the specified window's client area
nwSetWindowStyle	Changes the window style of the specified window
nwSetWindowText	Changes the caption string of the specified window

Nindows API

nwFindWindow	Window Support Function
Format	NHWND nwFindWindow(NHWND hWnd, Sint8* caption)
Parameters	HWnd- The parent window which starts searching for the window Caption- The caption string of the window being searched for
Return value	If the window is found, it returns its window handle else it returns NULL.
Function	<ul style="list-style-type: none"> • Searches the specified parent window's child windows for the window with the specified caption string. • If you want to search all windows, specify NULL for the parent window.
Reference	NwFindWindowByPos()
Note	
Example	<pre>//Searches all of the windows for the window "Material Window" hWnd = nwFindWindow(NULL, "Material Window");</pre>

nwFindWindowByPos	Window Support Function
Format	NHWND nwFindWindowByPos(Sint16 x, Sint16 y)
Parameters	x, y- Screen coordinates
Return value	If the window is found, it returns its window handle else it returns NULL.
Function	Searchs windows which are displayed in the specified screen coordinates (x, y).
Reference	nwFindWindow
Note	
Example	<pre>// Checks to see if the window is displayed at the coordinates of the mouse cursor NJS_PERIPHERAL* mouse = njGetPeripheral(NJD_PORT_SYSMOUSE); if (nwFindWindowByPos(mouse->x, mouse->y) { // The window is displayed } else { // The window is not displayed }</pre>

nwGetClientRect **Window Support Function**

Format Bool nwGetClientRect(NHWND hWnd, NWS_RECT* rect)

Parameters hWnd- Window handle
rect- Address which holds the rectangle information

Return value If it succeeds, it returns TRUE, else it returns FALSE

Function Gets the rectangle of the window's client area

Reference

Note • The rectangle it gets is the absolute coordinates on the screen without any relation to the NWD_WS_OFFSET flag in the window style.

Example

```
NWS_RECT rect;
nwGetClientRect(hWnd, &rect);
```

nwGetWindowColor **Window Support Function**

Format Bool nwGetWindowColor(NHWND hWnd, NWS_RGBA col[4])

Parameters hWnd- Window handle
col- Address of the NWS_RGBA structure array which gets the color

Return value If it succeeds, it returns TRUE, else it returns FALSE

Function Gets the window color. The color of the upper left vertex, upper right, lower right and lower left are stored in order from col[0].

Reference

Note

Example

```
NWS_RGBA col[4];
nwGetWindowColor(hWnd, col);
```

nwGetWindowPos	Window Support Function
Format	Bool nwGetWindowPos(NHWND hWnd, Sint32* x, Sint32* y)
Parameters	hWnd- Window handle x,y- Address which stores the coordinates
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Gets the upper left coordinates of the window's client area
Reference	
Note	<ul style="list-style-type: none">• If NWD_WS_OFFSET is specified in the window style, the coordinates are relative to the parent window.
Example	<pre>Sint32 x, y; nwGetWindowPos(hWnd, &x, &y);</pre>

nwGetWindowRect	Window Support Function
Format	Bool nwGetWindowRect(NHWND hWnd, NWS_RECT* rect)
Parameters	hWnd- Window handle rect- Address which stores the rectangle information
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Gets the window's entire rectangle, including the caption and border line
Reference	
Note	The rectangle it gets is the absolute coordinates on the screen without any relation to the NWD_WS_OFFSET flag in the window style.
Example	<pre>NWS_RECT rect; nwGetWindowRect(hWnd, &rect);</pre>

nwGetWindowSize **Window Support Function**

Format Bool nwGetWindowSize(NHWND hWnd, Sint32* w, Sint32* h)

Parameters hWnd- Window handle
 w,h- Address which stores the width and height

Return value If it succeeds, it returns TRUE, else it returns FALSE

Function Gets the width and height of the window's client area

Reference

Note

Example

```
Sint32 width, height;
nwGetWindowSize(hWnd, &width, &height);
```

nwGetWindowStyle **Window Support Function**

Format Bool nwGetWindowStyle(NHWND hWnd, Sint32* style)

Parameters hWnd- Window handle style- Address which gets the style

Return value If it succeeds, it returns TRUE, else it returns FALSE

Function Gets the window style

Reference

Note

Example

```
Sint32 style;
nwGetWindowStyle(hWnd, &style)
if (style & NWD_WS_SHADOW) {
    // if it is a window with a shadow
}
```

nwGetWindowText **Window Support Function**

Format	Sint32 nwGetWindowText(NHWND hWnd, Sint8* caption, Sint32 size)
Parameters	hWnd- Window handle caption- Buffer address which stores the window caption string size- Buffer size
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Gets the caption string displayed in the window's caption bar and copies it to the buffer
Reference	
Note	
Example	<pre>// Gets the window hWnd's in buf Sint8 buf[256]; nwGetWindowText(hWnd, buf, sizeof(buf));</pre>

nwSetWindowColor **Window Support Function**

Format	Bool nwSetWindowColor(NHWND hWnd, NWS_RGBA col[4])
Parameters	hWnd- Window handle col- Array address which stores the color of each window vertex
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Changes the window's color. Please specify the color of the upper left vertex, upper right, lower right and lower left in order from col[0].
Reference	
Note	If NWD_WS_SHADING is not specified in the window style, the color of the upper left vertex is applied to all vertices.
Example	<pre>NWS_RGBA col[4] = { {255, 0, 0,255}, // Color of the upper left vertex { 0,255, 0,255}, // Color of the upper right vertex { 0, 0,255,255}, // Color of the lower right vertex { 0, 0, 0,255}, // Color of the lower left vertex }; nwSetWindowColor(hWnd, col);</pre>

nwSetWindowPos	Window Support Function
Format	Bool nwSetWindowPos(NHWND hWnd, Sint32 x, Sint32 y)
Parameters	hWnd- Window handle x,y- Upper left coordinates of the client area
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Changes the display coordinates of the window. The window moves so that the upper left coordinates of the client area are at the point (x, y). If NWD_WS_OFFSET is specified in the window style, the coordinates are relative to the parent window.
Reference	
Note	
Example	<pre>// Moves the window to the point (100, 50). nwSetWindowPos(hWnd, 100, 50);</pre>

nwSetWindowSize	Window Support Function
Format	Bool nwSetWindowSize(NHWND hWnd, Sint32 w, Sint32 h)
Parameters	hWnd- Window handle w,h- Width and height of the client area
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Changes the size of the window. (w,h) are the width and height of the client area
Reference	
Note	
Example	<pre>// Changes the width and height of the window's client area to (128, 64) nwSetWindowSize(hWnd, 128, 64);</pre>

nwSetWindowStyle	Window Support Function
Format	Bool nwSetWindowStyle(NHWND hWnd, Sint32 and_style, Sint32 or_style)
Parameters	hWnd- Window handle and_style- and style or_style- or style
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Changes the window style
Reference	
Note	<ul style="list-style-type: none">• We cannot guarantee what will happen if the window class and other parameters have conflicting settings.• When using the and_style please do not forget to attach a " ~ " as in the example.• We cannot guarantee what will happen if the NWD_WS_CONTROL flag is set in this function for a window which did not have the NWD_WS_CONTROL flag set at the time it was created.
Example	<pre>// Removes the shadow from the window and attached a caption nwSetWindowStyle(hWnd, ~NWD_WS_SHADOW, NWD_WS_CAPTION);</pre>

NwSetWindowText	Window Support Function
Format	Bool nwSetWindowStyle(NHWND hWnd, Sint8* caption)
Parameters	HWnd- Window handle caption- Pointer to the NULL terminal caption string
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	Changes the caption displayed in the window's caption bar.
Reference	
Note	
Example	<pre>// Changes the window's caption to "New Caption" nwSetWindowText(hWnd, "New Caption");</pre>

Structure

NWS_WIN

Structure

Definition

```
typedef struct _NWS_WIN {
    Sint32 style;
    Sint32 wClass;
    Sint8 *caption;
    Sint32 font;
    struct _NWS_WIN *parent;
    struct _NWS_WIN *child;
    struct _NWS_WIN *before;
    struct _NWS_WIN *next;
    Sint32 x, y;
    Sint32 w, h;
    NWS_RGBA col[4];
    NWS_MSGHANDLE *msgHandle;
    void *menuTable;
    void *userBuf;
    void (*clientDraw)(struct _NWS_WIN *NWFUNC);
    void (*execFunc)(struct _NWS_WIN *NWFUNC);
    void (*destructor)(struct _NWS_WIN* NWFUNC);
    Sint32 param1, param2;
    struct _NWS_WIN* hClose;
    struct _NWS_WIN* hMaximize;
    struct _NWS_WIN* hMinimize;
} NWS_WIN;
```

Members

style- Window style
wClass- Window class
caption- Caption string
font- Font type
parent- Parent window handle
child- Child window handle
before- Previous window handle
next- Next window handle
x, y- Upper left coordinates of the client area
w, h- Width and height of the client area
col- Color of the 4 vertices
msgHandle- Not used (reserved)
menuTable- Menu table
userBuf- Buffer for user
clientDraw- Address of client drawing callback function
execFunc- Address of window execution function
destructor- Address of window destruction callback function
param1, param2- User parameters
hClose- Window handle of the close box
hMaximize- Reserved
hMinimize- Reserved

Description

The fundamental structure of all windows. The window handle is a pointer to this structure.

Reference

NWS_RGBA Structure

Definition	<pre>typedef struct _NWS_RGBA { Uint8 r; Uint8 g; Uint8 b; Uint8 a; } NWS_RGBA;</pre>
Description	It is mainly a structure that defines the window's color.
Members	r- Red(0-255) g- Green(0-255) b- Blue(0-255) a- Transparency(0-255) 0 is completely transparent, 255 is opaque
Reference	nwGetWindowColor(),nwSetWindowColor()

NWS_RECT Structure

Definition	<pre>typedef struct _NWS_RECT { Sint32 left; Sint32 top; Sint32 right; Sint32 bottom; } NWS_RECT;</pre>
Description	Structure which defines the rectangle area on the screen.
Members	left- Left side top- Top side right- Right side bottom- Bottom side
Reference	nwGetWindowRect()

5. Scroll Windows

5.1 Summary

A Scroll Window is a window which has the window class `NWD_WC_SCRWIN`.

Scroll Windows, unlike normal windows, have a function which allows scrolling of the contents displayed in the client area.

5.2 Creating a Scroll Window

A scroll window is created by using the function `nwCreateWindow()`, the same function used to create normal windows. A scroll window is created by specifying `NWD_WC_SCRWIN` in the window class.

```
static void scroll_window_callback(NHWND hWnd)
{
    nwTextOut(hWnd, 1, 1, "Scroll Window Sample");
}

Bool create_scroll_window(void)
{
    NHWND hWnd = nwCreateWindow(NWD_WC_SCRWIN,
                               "Scroll Window",
                               NWD_WS_CAPTION | NWD_WS_BORDER | NWD_WS_SHADOW,
                               50, 50, 100, 100,
                               NULL);

    if (!hWnd) return FALSE;

    hWnd->clientDraw = scroll_window_callback;
    return TRUE;
}

void njUserInit(void)
{
    :
    nwInitSystem(...);
    create_scroll_window();
}
```

A scroll window created in this manner looks like a normal window, but the client area can be scrolled with the mouse. Placing the mouse cursor in the client area and pressing the left button, the area can be freely scrolled by moving the mouse. Immediately after creating the window, the client area can be scrolled up and down and left and right, but it can also be set to only scroll up and down or only left and right.

Setting the window to only scroll up and down

```
nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL);
```

Setting the window to only scroll left and right

```
nwScrWinEnableScroll(hWnd, NWD_ES_HORIZONTAL);
```

Setting the window to scroll up and down as well as left and right

```
nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL | NWD_ES_HORIZONTAL);
```

Setting the window to not scroll in any direction

```
nwScrWinEnableScroll(hWnd, 0);
```

5.3 Description of Functions Used to Create a Scroll Window

Function	Description
nwScrWinEnableScroll	Enables or disables scrolling in the scroll window
nwScrWinSetClip	Sets the scrolling area of the scroll window
nwScrWinScroll	Scrolls the scroll window

Nindows API

nwScrWinEnableScroll	Scroll Window Function
Format	Bool nwScrWinEnableScroll(NWHWND hWnd, long flag)
Parameters	hWnd- Window handle of the scroll window flag- The direction in which you want to enable scrolling NWD_ES_VERTICAL Enable up/ down scrolling NWD_ES_HORIZONTAL Enable left/ right scrolling
Return value	If it succeeds, it returns TRUE, else it returns FALSE
Function	<ul style="list-style-type: none"> • Sets the scrolling direction of the scroll window's client area • Set the flag to 0 to disable scrolling in any direction • Set the flag to 1 to enable scrolling up/ down and left/ right.
Reference	NwScrWinSetClip(),nwScrWinScroll()
Note	<ul style="list-style-type: none"> • This function's settings are only valid for scrolling with the mouse. Scrolling via nwScrWinScroll() is always valid in every direction.
Example	<pre>// Enables scrolling in every direction nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL NWD_ES_HORIZONTAL);</pre>

nwScrWinSetClip Scroll Window Function

Format	Bool nwScrWinSetClip(NHWND hWnd, NWS_RECT* rect)
Parameters	hWnd- Window handle of the scroll window rect -Rectangular area with scrolling enabled
Return value	If it succeeds, it returns TRUE, else it returns FALSE__
Function	<ul style="list-style-type: none">• Sets the range of scrolling in the client area of the scroll window.
Reference	NwScrWinEnableScroll(),nwScrWinScroll()
Note	<ul style="list-style-type: none">• This function's settings are only valid for scrolling with the mouse.• The initial value of the clipping area is an area nine (3x3) times the size of the client area, centered on the client area.
Example	<pre>// Sets the range of scrolling to (-10,-10)-(10,10) NWS_RECT rect = {-10, -10, 10, 10}; NwScrWinSetClip(hWnd, &rect);</pre>

nwScrWinScroll Scroll Window Function

Format	Bool nwScrWinScroll(NHWND hWnd, Sint32 x, Sint32 y)
Parameters	HWnd- Window handle of the scroll window x, y- Scrolling value
Return value	If it succeeds, it returns TRUE, else it returns FALSE__
Function	Scrolls the client area of the scroll window by the specified number of dots
Reference	NwScrWinEnableScroll(),nwScrWinSetClip()
Note	
Example	<pre>//Scrolls up one dot at a time Sint32 njUserMain(void) { NHWND hWnd= nwFindWindow(NULL, "Scroll Window"); if (hWnd) { nwScrWinScroll(hWnd, 0, 1); } : : return nwExecute(); }</pre>

nwScrWinGetScroll Scroll Window Function

Format Bool nwScrWinGetScroll(NHWND hWnd, Sint32* x, Sint32* y)

Parameters hWnd- Window handle of the scroll window
x, y Poi-nter which gets the scroll coordinates

Return value If it succeeds, it returns TRUE, else it returns FALSE__

Function Gets the present scrolling coordinates of the scroll window

Reference nwScrWinScroll()

Note

Example

```
//Sets scrolling Sint32 x, y;
nwScrWinGetScroll(hWnd, &x, &y);
nwScrWinSetScroll(hWnd, -x, -y);
```

6. Edit Windows

6.1 Summary

An Edit Window is a window which has the window class `NWD_WC_EDITWIN` and it includes the functions of a scroll window(window class `NWD_WC_SCRWIN`).

An edit window has the following special features.

- It has a text buffer, text can be set and automatically displayed.
- Several lines of text can be displayed.
- The window's client area can also be made to scroll.

The Nindows utility "Debug Window" is created as an Edit Window.

6.2 Creating and Using an Edit Window

An Edit Window is created by calling the function `nwCreateEditWindow()`.

```
NHWND hWnd;  
  
hWnd = nwCreateEditWindow(500, "Edit Window 1",  
    NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_THICKFRAME | NWD_WS_SHADING,  
    50, 50, 150, 100, NULL);
```

Next, let's add some text to this window. This is accomplished by using the function `nwEditWinAddString()`.

```
nwEditWinAddString(hWnd, "ABCD\n");  
nwEditWinAddString(hWnd, "0123\n");
```

The function `nwEditWinPrintf()` is also available to use in the same way as the `printf()` function.

```
nwEditWinPrintf(hWnd, "ADDRESS : %08X\n", 0);
```

As a result of the preceding operations, the following window will be displayed on the screen.

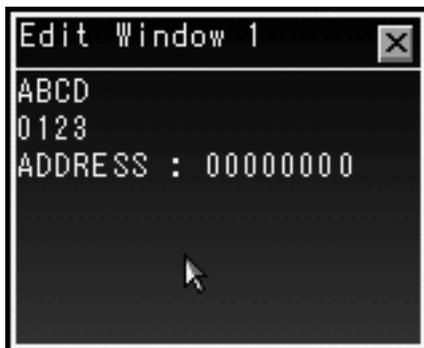


Diagram 6-1. An example of creating an Edit Window

As text strings are added to the Edit Window and they cannot be displayed in the window's client area, the window will automatically scroll.

Furthermore, the Edit Window also has the functionality of a Scroll Window, so the display contents of the client area can be freely scrolled by dragging the mouse.

When the added text fills up the buffer, data will be erased from the beginning of the buffer.

An Edit Window is destroyed like any other window by using the function. `nwDestroyWindow()`.

```
nwDestroyWindow(hWnd);
```

*In the present version of Nindows, text strings set using `nwEditWinAddString()` and `nwEditWinPrintf()` must have the linefeed character '\n' at the end. Please note that if the linefeed character is not appended, the previously entered text will not be displayed until text which includes the linefeed character is set using these functions.

6.3 Description of Functions Used in Creating Edit Windows

Function	Description
NwCreateEditWindow	Creates an Edit Window
NwEditWinAddString	Adds text to an Edit Window
NwEditWinPrintf	Adds text to the Edit Window in the printf() format

Windows API

nwCreateEditWindow	Window Creation Function
Format	NHWND nwCreateEditWindow(Sint32 lines, Sint8* caption, Sint32 style, Sint32 x, Sint32 y, Sint32 w, Sint32 h, NHWND hWndParent)
Parameters	lines- Maximum number of text lines caption- Window name string (caption) style- Window style x,y- Upper left coordinate of the client area w,h- Width and height of the client area hWndParent- Parent window handle
Return value	If successful, it returns the handle of the created edit window, or if it couldn't create an edit window it returns NULL.
Function	Creates an edit window and reserves a text buffer.
Reference	nwDestroyWindow(),nwCreateMenuWindow(),nwCreateEditWindow(), nwCreateScrollBar(),nwCreateButton() NWS_WIN structure
Note	
Example	<pre>NHWND hWnd; hWnd = nwCreateEditWindow(500, "Edit Window", NWD_WS_CAPTION NWD_WS_CONTROL NWD_WS_BORDER NWD_WS_SHADING, 50, 50, 150, 100, NULL);</pre>

nwEditWinAddString	Edit Window Function
Format	Bool nwEditWinAddString(NHWND hWnd, Sint8* string)
Parameters	hWnd- Window handle of the edit window that text is being added to string- Pointer to the text being added
Return value	If the text is successfully added, it returns TRUE, else it returns FALSE.
Function	Adds text to an edit window and displays it.
Reference	nwCreateEditWindow(),nwEditWinPrintf()
Note	
Example	<pre>nwEditWinAddString(hWnd, "AddText\n");</pre>

nwEditWinPrintf	Edit Window Function
Format	Bool nwEditWinPrintf(NWHWND hWnd, Sint8* fmt, ...)
Parameters	hWnd- Window handle of the edit window that text is being added to printf() format fmt-text string
Return value	If the text is successfully added, it returns TRUE, else it returns FALSE.
Function	<ul style="list-style-type: none">• Adds text to an edit window and displays it.• The printf() format can be used.
Reference	NwCreateEditWindow(),nwEditWinAddString()
Note	
Example	NwEditWinPrintf(hWnd, "i = %d\n", i);

6.4 Description of Functions Used in Nindows' Debug Window Utility

nwDebugPrintf	Edit Window Function
Format	void nwDebugPrintf(Sint8* fmt, ...)
Parameters	fmt- printf() format text string
Return value	None
Function	<ul style="list-style-type: none">• Adds text to the Debug window and displays it• The printf() format can be used.
Reference	NwEditWinPrintf()
Note	
Example	NwDebugPrintf("i = %d\n", i);

7. Scrollbar Controls

7.1 Summary

Scrollbars are controls which are very well suited to adjusting various numerical parameters. For example, they can be used to change things such as backgrounds, the color of models, adjusting the movement speed of objects, and various other uses.

7.2 Creating Scrollbar Controls

This example shows how to change a model's material (NJS_ARGB structure) using a scrollbar.

```
static NJS_ARGB argb = { 0.f, 0.f, 0.f, 0.f};
static NWS_SCROLLBARINFO material_scroll_info[] = {
    /* caption      data          min      max      line page  pos */
    {"Alpha Scroll", {&argb.a, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Red Scroll",   {&argb.r, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Green Scroll", {&argb.g, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
    {"Blue Scroll",  {&argb.b, NWD_DT_FLOAT}, -256.f, 255.f, 1.f, 10.f, 0.f},
};
static NWS_SCROLLBARLIST material_scroll_list = {
    sizeof(material_scroll_info) / sizeof(NWS_SCROLLBARINFO),
    NWD_WS_SB_HORZ, 50, 3, 200, 14,
    material_scroll_info,
};
static void draw_material_window_callback(NHWND hWnd)
{
    nwTextOut(hWnd, 2, 2, "A %3.0f", argb.a);
    nwTextOut(hWnd, 2, 16, "R %3.0f", argb.r);
    nwTextOut(hWnd, 2, 30, "G %3.0f", argb.g);
    nwTextOut(hWnd, 2, 44, "B %3.0f", argb.b);
}
static void create_material_window(void)
{
    NHWND hWnd = nwCreateWindow(NWD_WC_WIN, "Material Window",
                               NWD_WS_CAPTION | NWD_WS_SHADING |
                               NWD_WS_BORDER | NWD_WS_CONTROL,
                               40, 382, 260, 64, NULL);
    hWnd->clientDraw = draw_material_window_callback;
    nwCreateScrollBarArray(&material_scroll_list, hWnd);
}
void njUserInit(void)
{
    :
    create_material_window();
}
```

```

Sint32 njUserMain(void)
{
    :
    :
    njSetConstantMaterial (&argb);
    njDrawObject (OBJECT);
    return nwExecute();
}

```

In this example, we first create a parent window called "Material Window" and then four scrollbars as its child windows. This is the standard way to do it. When this is done, the following window is displayed.

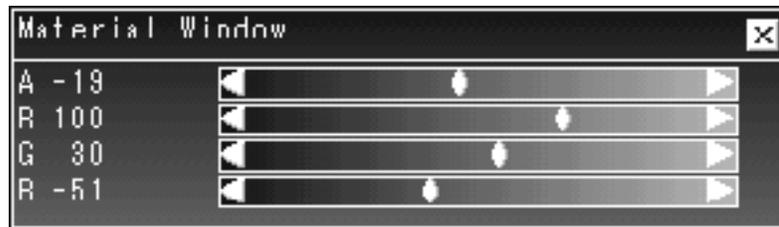


Diagram 7-1 . An example of creating a scrollbar control

The values of the NJS_ARGB structure members a,r,g,b change in response to manipulation of the scrollbar's knob. When the "Material Window" is destroyed, its four child windows, the scrollbars, are automatically destroyed.

7.3 Description of Functions Used in Creating Scrollbar Controls

Function	Description
NwCreateScrollBarArray	Creates several scrollbar control together

Windows API

nwCreateScrollBarArray	Scrollbar Function
Format	Bool nwCreateScrollBarArray(NWS_SCROLLBARLIST* list, NWHWND hWndParent)
Parameters	list- Pointer to the scrollbar list
hWndParent- Parent window handle	
Return value	If all the scrollbar controls were created, it returns TRUE, else it returns FALSE.
Function	Creates several scrollbar controls together and sets the parameters.
Reference	Low-level Scrollbar Function: nwCreateScrollBar(),nwSetScrollBarPos(),nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarLineMove(),nwSetScrollBarPageMove()
Note	This function is easier to use than creating scrollbars one by one and setting their parameters, but you cannot get the handles of the scrollbars that were created. Usually there is no need to get the scrollbar handles, but if necessary please use a low level scrollbar function or search for the window using nwFindWindow().
Example	

Structure

NWS_SCROLLBARLIST	Structure
Definition	<pre>typedef struct { Sint32 n; Sint32 style; Sint16 x, y; Sint16 w, h; NWS_SCROLLBARINFO* info; } NWS_SCROLLBARLIST;</pre>
Description	When creating scrollbars with the function nwCreateScrollBarArray(), it is needed with the NWS_SCROLLBARINFO structure.
Members	n- Number of elements in the NWS_SCROLLBARINFO array style- NWD_WS_SB_HORZ for a horizontal scrollbar, NWD_WS_SB_VERT for a vertical scrollbar. x, y- Display coordinates of the first scrollbar (relative to parent window). w, h- Width and height of one scrollbar. info- Array address of the NWS_SCROLLBARINFO structure
Reference	NWS_SCROLLBARINFO,NWS_DATA, nwCreateScrollBarArray()

NWS_SCROLLBARINFO Structure

Definition typedef struct { Sint8* caption; NWS_DATA data; Float min, max; Float line, page; Float pos; } NWS_SCROLLBARINFO;

Description When creating scrollbars with the function nwCreateScrollBarArray(), it is needed with the NWS_SCROLLBARLIST structure. One of these structures corresponds to one scrollbar. Usually used as an array to create several scrollbars together.

Members caption- Scrollbar caption strings.
 data- Pointer to the data structure associated with the scrollbar
 min- The minimum value in the associated data.
 max- The maximum value in the associated data.
 line- The amount of data changed when the scrollbar's arrow is clicked.
 page- The amount of data changed when the scroll area is clicked.
 pos- Initial value of the associated data.

Reference NWS_SCROLLBARLIST,NWS_DATA, nwCreateScrollBarArray()

NWS_DATA Structure

Definition typedef struct _NWS_DATA { void *dt; int type; } NWS_DATA;

Description Data structure associated with a scrollbar

Members dt- Pointer to the data.
 type- Data type. Specify from the table below.

Reference Low-level Scrollbar Function

Data Type	Meaning
NWD_DT_CHAR	char(Sint8) data type
NWD_DT_SHORT	short(Sint16) data type
NWD_DT_LONG	long(Sint32) data type
NWD_DT_FLOAT	float(Float) data type
NWD_DT_UCHAR	unsigned char(Uint8) data type
NWD_DT_USHORT	unsigned short(Uint16) data type
NWD_DT_ULONG	unsigned long(Uint32) data type

7.4 Creating Scrollbar Controls that Use Low-level Scrollbar Functions

Here we will discuss how to create the same material window using a more low-level function than the previously described `nwCreateScrollBarArray()`. Those readers who are not interested in this example may skip to the next chapter.

```
static NJS_ARGB argb = { 0.f, 0.f, 0.f, 0.f};
static NWS_DATA dt[] = {
    {(void*)&argb.a, NWD_DT_FLOAT},
    {(void*)&argb.r, NWD_DT_FLOAT},
    {(void*)&argb.g, NWD_DT_FLOAT},
    {(void*)&argb.b, NWD_DT_FLOAT},
};
static void draw_material_window_callback(NHWND hWnd)
{
    nwTextOut(hWnd, 2, 2, "A %3.0f", argb.a);
    nwTextOut(hWnd, 2, 16, "R %3.0f", argb.r);
    nwTextOut(hWnd, 2, 30, "G %3.0f", argb.g);
    nwTextOut(hWnd, 2, 44, "B %3.0f", argb.b);
}
static void create_material_window(void)
{
    NHWND hSc1;
    NHWND hWnd = nwCreateWindow(NWD_WC_WIN, "Material Window",
        NWD_WS_CAPTION | NWD_WS_SHADING |
        NWD_WS_BORDER | NWD_WS_CONTROL,
        40, 382, 260, 64, NULL);
    hWnd->clientDraw = draw_material_window_callback;
    hSc1 = nwCreateScrollBar(NWD_WS_SB_HORZ, "Alpha Scroll", 80, 3, 200, 11, hWnd);
    nwSetScrollBarData(hSc1, &dt[0]);
    nwSetScrollBarRange(hSc1, -256.0f, 255.0f);
    nwSetScrollBarPos(hSc1, 0.f);
    nwSetScrollBarLineMove(hSc1, 1.f);
    nwSetScrollBarPageMove(hSc1, 10.f);
    hSc1 = nwCreateScrollBar(NWD_WS_SB_HORZ, "Red Scroll", 80, 17, 200, 25, hWnd);
    nwSetScrollBarData(hSc1, &dt[1]);
    nwSetScrollBarRange(hSc1, -256.0f, 255.0f);
    nwSetScrollBarLineMove(hSc1, 1.f);
    nwSetScrollBarPageMove(hSc1, 10.f);
    nwSetScrollBarPos(hSc1, 0.f);
    hSc1 = nwCreateScrollBar(NWD_WS_SB_HORZ, "Green Scroll", 80, 31, 200, 39, hWnd);
    nwSetScrollBarData(hSc1, &dt[2]);
    nwSetScrollBarRange(hSc1, -256.0f, 255.0f);
    nwSetScrollBarPos(hSc1, 0.f);
    nwSetScrollBarLineMove(hSc1, 1.f);
    nwSetScrollBarPageMove(hSc1, 10.f);
}
```

The code above will create a "Material Window" that looks and functions the same as the one in Diagram 7-1.

7.5 Description of Low-level Scrollbar Functions

Function	Description
NwCreateScrollBar	Creates a scrollbar control
NwSetScrollBarData	Associates data with a scrollbar
NwSetScrollBarRange	Sets the extent of the scrollbar
NwSetScrollBarPos	Sets the position of the scrollbar knob
NwSetScrollBarLineMove	Sets the distance to move when the scrollbar's arrow is clicked
NwSetScrollBarPageMove	Sets the distance to move when the scrollbar's area is clicked.

Table 7-3. List of Low-level Scrollbar Functions

Nindows API

nwCreateScrollBar	Low-level Scrollbar Function
Format	NHWND nwCreateScrollBar(Sint32 type, Sint8 *caption, Sint32 x, Sint32 y, Sint32 w, Sint32 h, NHWND hWndParent);
Parameters	type- Horizontal scrollbars are NWD_WS_SB_HORZ, vertical scrollbars are NWD_WS_SB_VERT caption- Scrollbar caption string x, y- Coordinates of scrollbar creation (relative to parent) w, h- Scrollbar width and height hWndParent- Window handle of the parent window
Return value	If successful in creating the scrollbar controls, it returns the window handle of the created scrollbar controls, else it returns NULL.
Function	Creates scrollbar controls.
Reference	nwCreateScrollBarArray(), Low-level Scrollbar Function: nwSetScrollBarPos(),nwSetScrollBarRange(),nwSetScrollBarData(), nwSetScrollBarLineMove(),nwSetScrollBarPageMove()
Note	Please initialize the settings of the scrollbars created using nwSetScrollBarPos(),nwSetScrollBarRange(),nwSetScrollBarData().
Example	<pre>NHWND hSc1 = nwCreateScrollBar(NWD_WS_SB_HORZ, "Alpha Scroll", 80, 3, 200, 11, hWndParent);</pre>

nwSetScrollBarData	Low-level Scrollbar Function
Format	<code>void nwSetScrollBarData(NHWND hScI, NWS_DATA * data)</code>
Parameters	hScI- Window handle of the scrollbar control data- Associated data structure
Return value	None
Function	Associates data with the scrollbar control
Reference	<code>nwSetScrollBarPos()</code> , <code>nwSetScrollBarRange()</code> , <code>nwSetScrollBarLineMove()</code> , <code>nwSetScrollBarPageMove()</code>
Note	
Example	<pre>// Associates the long variable a with the scrollbar long a; NWS_DATA data = {&a, NWD_DT_LONG}; nwSetScrollBarData(hWnd, &data);</pre>

nwSetScrollBarRange	Low-level Scrollbar Function
Format	<code>void nwSetScrollBarRange(NHWND hScI, Float min, Float max)</code>
Parameters	hScI- Window handle of the scrollbar control min- Minimum value of the data associated with the scrollbar max- Maximum value of the data associated with the scrollbar
Return value	None
Function	Sets the range of the scrollbar
Reference	<code>nwSetScrollBarPos()</code> , <code>nwSetScrollBarData()</code> , <code>nwSetScrollBarLineMove()</code> , <code>nwSetScrollBarPageMove()</code>
Note	
Example	<pre>// Sets the range to (-30 ~ 30) nwSetScrollBarRange(hWnd, -30.f, 30.f);</pre>

nwSetScrollBarPos	Low-level Scrollbar Function
Format	void nwSetScrollBarPos(NHWND hScI, Float pos)
Parameters	hScI- Window handle of the scrollbar control pos- Value of the data associated with the scrollbar
Return value	None
Function	Sets the value of the data associated with the scrollbar
Reference	nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarLineMove(),nwSetScrollBarPageMove()
Note	Used when setting the initial data value when the scrollbar is created.
Example	<pre>// Sets the initial data value to 0 nwSetScrollBarPos(hWnd, 0.f);</pre>

nwSetScrollBarLineMove	Low-level Scrollbar Function
Format	void nwSetScrollBarLineMove(NHWND hScI, Float step)
Parameters	hScI- Window handle of the scrollbar control step- Step value
Return value	None
Function	Sets the amount of data changed when the scrollbar arrows are pressed
Reference	nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarPos(),nwSetScrollBarPageMove()
Note	The default is 1.0
Example	<pre>// Sets the amount of data changed when the scrollbar arrow is pressed to 2 nwSetScrollBarLineMove(2.f);</pre>

nwSetScrollBarPageMove **Low-level Scrollbar Function**

Format	void nwSetScrollBarPageMove(NHWND hScI, Float step)
Parameters	hScI- Window handle of the scrollbar control step- Step value
Return value	None
Function	Sets the amount of data changed when the scrollbar's area is clicked
Reference	nwSetScrollBarRange(),nwSetScrollBarData(),nwSetScrollBarPos(),nwSetScrollBarLineMove()
Note	The default is 10.0
Example	<pre>// Sets the amount of data changed when the scrollbar area is clicked to 5 nwSetScrollBarLineMove(5.f);</pre>

8. Button Controls

8.1 Summary

A button is a control which produces a callback when clicked and can be used for many purposes. Buttons are convenient for many uses such as a toggle switch for application flag variables, an interface for choosing one object out of a group, etc.

8.2 Creating a Button Control

As an example, we will create a sample which selects the textures used in environment mapping by using "Back" and "Next" buttons.

```

#define TEXTURES 4
static int texno = 0;
static void test_window_callback(NHWND hWnd)
{
    char buf[256];
    sprintf(buf, "Texture=%d", texno);
    nwTextOut(hWnd, 1, 1, buf);
}
static void button_callback_back(NHWND hWnd)
{
    texno--;
    if (texno < 0) texno = TEXTURES - 1;
}
static void button_callback_next(NHWND hWnd)
{
    texno++;
    if (texno >= TEXTURES) texno = 0;
}
Bool create_test_window(void)
{
    NHWND hWndParent;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Texture Select",
                                NWD_WS_CAPTION | NWD_WS_BORDER |
                                NWD_WS_SHADOW,
                                50, 50, 128, 48,
                                NULL);

    if (!hWndParent) return FALSE;
    hWndParent->clientDraw = test_window_callback;
    nwCreateButton(button_callback_back, "Back", 3, 20, 48, 13, hWndParent);
    nwCreateButton(button_callback_next, "Next", 56, 20, 48, 13, hWndParent);
    return TRUE;
}
 Sint32 njUserMain(void)

```

```
{
    :
    :
    njSetTexture(&texlist[texno]);
    njDrawObject(OBJECT);
    return nwExecute();
}
```

As a result of the previous code, the following texture selection window is displayed.



Diagram 8-1. Example Creation of a Texture Selection Window

When the "Back" and "Next" buttons are clicked, the specified callback functions are called, and the value of the variable `texno` is changed. In response, the texture used in the environment mapping also changes.

When the texture selection window is destroyed, its child windows, the two buttons, are also automatically destroyed.

8.3 Button Validity and Invalidity

In the preceding example, the "Back" and "Next" buttons are always valid, a callback will always work when they are clicked. However, depending on the situation, there is a need to do things like disable a button. Let's modify the previous sample to add that kind of operation.

The operation to be added will make the "Back" button invalid when the buttons are created and check the texture number in the button's parent window callback function to set the two buttons to valid or invalid. In order to do this, we should make the buttons' window handles into global variables.

To set the buttons to valid or invalid, we will use the function `nwEnableButton()`.

Operation to make the button valid

```
nwEnableButton(button, TRUE);
```

Operation to make the button invalid

```
nwEnableButton(button, FALSE);
```

The text on an invalid button is displayed with a light color and even if the button is clicked, the animation and callback will not work.

```
#define TEXTURES 4
static int texno = 0;
static NWHWND button_back;
static NWHWND button_next;
static void test_window_callback(NWHWND hWnd)
{
```

```

char buf[256];
sprintf(buf, "Texture=%d", texno);
nwTextOut(hWnd, 1, 1, buf);
// If the texture number is 0, the "Back" button is disabled
if (texno == 0) nwEnableButton(button_back, FALSE);
else nwEnableButton(button_back, TRUE);
// If the texture number is 3, the "Next" button is disabled
if (texno == TEXTURES - 1) nwEnableButton(button_next, FALSE);
else nwEnableButton(button_next, TRUE);
}
static void button_callback_back(NHHWND hWnd)
{
    texno--;
    if (texno < 0) texno = 0;
}
static void button_callback_next(NHHWND hWnd)
{
    texno++;
    if (texno >= TEXTURES) texno = TEXTURES - 1;
}
Bool create_test_window(void)
{
    NHHWND hWndParent;
    hWndParent = nwCreateWindow(NWD_WC_WIN,
                                "Texture Select",
                                NWD_WS_CAPTION | NWD_WS_BORDER |
                                NWD_WS_SHADOW,
                                50, 50, 128, 48,
                                NULL);

    if (!hWndParent) return FALSE;
    hWndParent->clientDraw = test_window_callback;
    button_back = nwCreateButton(button_callback_back,
                                "Back", 3, 20, 48, 13, hWndParent);
    button_next = nwCreateButton(button_callback_next,
                                "Next", 56, 20, 48, 13, hWndParent);
    // At the start, the texture number is 0, so the "Back" button is disabled
    nwEnableButton(button_back, FALSE);
    return TRUE;
}

```

8.4 Description of Functions for Button Controls

Function	Description
nwCreateButton	Creates a button control
nwEnableButton	Switches a button's validity and invalidity

Windows API

nwCreateButton	Button Function
Format	NHWND nwCreateButton(NWF_BUTTONFUNC func, Sint8 *caption, Sint32 x, Sint32 y, Sint32 w, Sint32 h, HWND hWndParent);
Parameters	func- Button callback function caption- Text displayed on the button surface x, y- Coordinates where the button is created (relative to parent) w, h- Width and height of button hWndParent- Window handle of parent window
Return value	If the button creation is successful, it returns the window handle, else it returns NULL.
Function	Creates button controls.
Reference	nwEnableButton(),nwDestroyWindow(),
Note	A button which has just been created is valid.
Example	<pre>//Creates an "OK" button HWND button = nwCreateButton(button_callback_back, "OK", 3, 20, 48, 13, hWndParent);</pre>

nwEnableButton	Button Function
Format	void nwEnableButton(HWND hWnd, Bool flag)
Parameters	hWnd- Button's window handle flag- If the button is valid it is TRUE, else FALSE
Return value	None
Function	Sets a button to valid or invalid. An invalid button has text displayed in a light color and will not work even if clicked.
Reference	nwCreateButton()
Note	A button which has just been created is valid.
Example	<pre>// Make a button invalid nwEnableButton(button, FALSE);</pre>

Callback Function

ButtonCallback	Button Callback Function
Format	void ButtonCallback(NHWND hWnd)
Parameters	hWnd- Button handle where the callback originated
Return value	None
Function	Application defined function which is called back when the button is clicked.
Reference	nwCreateButton(),nwEnableButton()
Note	

9. Menus

9.1 Summary

Nindows has an API for creating popup menus like common GUI systems.

The most representative menu in Nindows is the System Menu, but inside this menu is an item labeled "User (undefined)" in light colored text.



Diagram 9-1. The "User (undefined)" item in the System Menu

This menu item is for setting user defined menus. By creating a menu table and entering it into this item, user defined menus can be easily used. This chapter discusses how to create and enter menu tables. It will also cover how to create windows that popup directly without entering them in the System Menu.

9.2 Creating and Entering Menu Tables

Menu tables are created as an array of NWS_MENUTABLE. The following is an example of the simplest menu table with one item.

```
static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}
static NWS_MENUTABLE menu_table[] = {
    // Item 1
    {
        item                NWD_MF_NORMAL,                // Flag showing it is a normal menu
                            "Test Menu 1",                //menu item string
                            menu_callback,                // Callback function called when the
item is selected
                            0,                            // Parameter passed to the callback
        function
    },
    // End of table
    {
        NWD_MF_NULL,                //End of the table
        "",
        NULL,
        0
    },
};
```

To enter this menu table into the System Menu's "User(undefined)" item, we will use the function `nwSetUserMenu()`.

```
void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    nwSetUserMenu(test_menu);
}
```

By calling this function the light colored "User(undefined)" item has changed to "User >" and the "Test Menu 1" which was entered pops up as a sub-menu.



Diagram 9-2. Condition where the user menu has been entered (1)

When this menu is selected, the callback function `menu_callback()` set in the menu table is called back. `menu_callback()` isn't doing any processing, so nothing happens. This callback function will be explained in the next section.

Let's look at a more complex example of a menu table.

```
static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE menu_table[] = {
    // Item 1
    {
        NWD_MF_NORMAL,           // Flag showing it is a normal menu item
        "Test Menu 1",          // Menu item string
        menu_callback,          // Callback function called when the item is selected
        0,                       // Parameter passed to the callback function
    },
    // Item 2
    {
        NWD_MF_SEPARATOR,       // Separator
        "",
        NULL,
        0,
    }
    // Item 3
    {
        NWD_MF_POPUP,           // Flag showing that it is a popup menu
        "Test Popup 1",         // Menu item string
        NULL,
        (Sint32)submenu_table, // Menu table address of the sub menu
    }
}
```

```
    },
    // End of table
    {
        NWD_MF_NULL, // End of the table
        "",
        NULL,
        0,
    }
};

static void submenu_callback(NHWND hWndMenu, Sint32 idx, Sint32 param)
{
}

static NWS_MENUTABLE submenu_table[] = {
    // Item 1
    {
        NWD_MF_NORMAL | NWD_MF_GRAYED, // Normal menu item, cannot be selected
        "Sub Menu 1", // Menu item string
        submenu_callback, // Callback function
        0, // Parameter passed to the callback function
    },
    // Item 2
    {
        NWD_MF_NULL, // End of the table
        "",
        NULL,
        0
    },
};
```

If this table is entered in the same way using `nwSetUserMenu()`, you get the following menu.

Furthermore, when a new menu is entered using `nwSetUserMenu()`, the previously entered menu table is overwritten and the new menu is enabled.

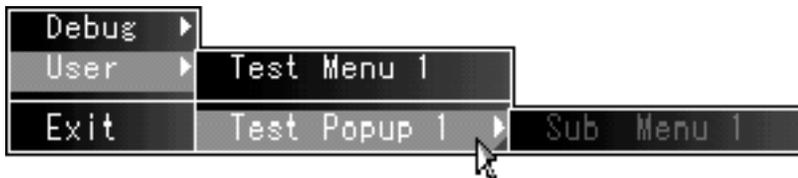


Diagram 9-3. Condition where the user menu has been entered (2)

The entry of a user menu is deleted in the following way.

```
nwSetUserMenu(NULL);
```

Once again, the display changes to a lightly colored "User(undefined)" and the user menu cannot be selected.

9.3 Menu Callback Functions

Menu callback functions are user defined functions, entered in the menu table, which are called back when the menu is selected. In the previous example the function `menu_callback()` was a callback function.

```
static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
```

The window handle of the menu window where the callback originated is passed to `hWndMenu`. There is usually no need to do this.

The parameter `idx` is the numerical position of the selected menu item in the menu, starting from 0. This can be used to tell which menu item has been selected in such cases where you want to process several menu items with the same callback function.

`param` is a parameter defined by the user in the menu table. In the same way, this is used when you want to process several menu items with one callback function.

9.4 Checkmarks

Checkmarks can be displayed on the left side of the menu item text strings. Checkmarks are useful for telling the user if an item is valid or if it is being selected. The diagram shows the Nindows utility "Ninja Info" when it is selected. The checkmark to the left of the "Ninja Info" item name shows that the "Ninja Info" window is being displayed. If the "Ninja Info" window is closed, the checkmark will disappear.

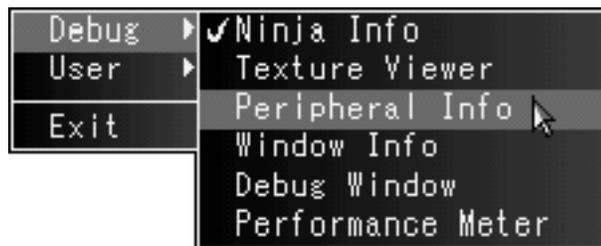


Diagram 9-4. Checkmark Example

The display of the checkmarks is turned on and off by directly setting the `type` member in the menu table. In the example, the checkmark for the top item in the menu table called `menu_table` is switched.

Display checkmark

```
menu_table[0].type |= NWD_MF_CHECKED;
```

Hide checkmark

```
menu_table[0].type &= ~NWD_MF_CHECKED;
```

Let's look at a more concrete example.

```
static NWS_MENUTABLE menu_table[] = {
    {NWD_MF_NORMAL, "Test Window", create_test_window, 0},
    {NWD_MF_NULL, "", NULL, 0},
};
static void destroy_test_window_callback(NWHWND hWnd)
{
    menu_table[0].type &= ~NWD_MF_CHECKED;
}
static void create_test_window(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
    NWHWND hWnd;
    // If a window is already created, it is destroyed
    // Checkmarks are erased by the window destructor
    if (hWnd = nwFindWindow(NULL, "Test Window")) {
        nwDestroyWindow(hWnd);
    } else {
        hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window",
            NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_SHADING |
NWD_WS_BORDER,
            50, 50, 100, 100, NULL);
        // Setting the destructor
        hWndNjInfo->destructor = destroy_test_window_callback;
        // "Test Window 1" was created, so a checkmark is placed on the menu
item
        menu_table[0].type |= NWD_MF_CHECKED;
    }
}
void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    nwSetUserMenu(menu_table);
}
```

This menu table is entered with `nwSetUserMenu()` and when `Test Window` is selected, it performs the window creation and destruction and then switches the checkmarks.

The reason why the window destructor (`hWnd->destructor`) is set and inside that the checkmarks are erased is because there are cases where windows are destroyed by methods other than menu selection. The following code looks correct, but in cases such as when the close box is clicked and the window is destroyed, the checkmark is not erased.

```

static void create_test_window(NHWND hWndMenu, Sint32 idx, Sint32 param)
{
    NHWND hWnd;
    if (hWnd = nwFindWindow(NULL, "Test Window")) {
        nwDestroyWindow(hWnd);
        menu_table[0].type &= ~NWD_MF_CHECKED;
    } else {
        hWnd = nwCreateWindow(NWD_WC_WIN, "Test Window",
            NWD_WS_CAPTION | NWD_WS_CONTROL | NWD_WS_SHADING |
NWD_WS_BORDER,
            50, 50, 100, 100, NULL);
        menu_table[0].type |= NWD_MF_CHECKED;
    }
}

```

9.5 Description of Functions for Entering User Menus

Function	Description
nwSetUserMenu	Enters a user menu in the System Menu

Nindows API

nwSetUserMenu	Menu Function
Format	void nwSetUserMenu(NWS_MENUTABLE* menuTbl)
Parameters	menuTbl- Array address of the menu table structure
Return value	None
Function	Enters user menus as popup menus in the "User" item of the System Menu. If the argument is specified as NULL, the previously entered menu is destroyed.
Reference	
Note	
Example	nwSetUserMenu(user_menu);

Callback Function

MenuCallback	Menu Function
Format	void MenuCallback(NHWND hWnd, Sint32 idx, Sint32 param)
Parameters	hWnd- Window handle of the menu window where the callback originated idx- Index of the selected menu item in the menu table param- Parameter set in the menu table
Return value	None
Function	User defined function called back by the menu window when the menu is selected.
Reference	
Note	
Example	

Structure

NWS_MENUTABLE	Structure
Definition	typedef struct _NWS_MENUTABLE { Sint32 type; Sint8 *title; NWF_MENUHANDLE func; Sint32 param; } NWS_MENUTABLE;
Description	Defines the contents of the menu when a user menu is entered with the nwSetUserMenu() function or when a menu window is created with the nwCreateMenu() function.
Members	type- Menu item type title- Menu item text func- Callback function for when the menu is selected param- Parameter passed to the callback function
Reference	

Menu Type Flag	Meaning
NWD_MF_NORMAL	Normal menu item. Cannot be specified at the same time with NWD_MF_POPUP, NWD_MF_SEPARATOR.
NWD_MF_POPUP	Has a popup sub-menu. Cannot be specified at the same time with NWD_MF_NORMAL, NWD_MF_SEPARATOR
NWD_MF_SEPARATOR	Separator. Cannot be specified at the same time with NWD_MF_NORMAL, NWD_MF_POPUP.
NWD_MF_CHECKED	Has a checkmark.
NWD_MF_GRAYED	Item displayed with a light color, cannot be selected.

9.6 Creating Popup Menus

9.6.1 Creating a Simple Popup Menu

Up until now, we have discussed how to set a user menu in the System Menu, but there is also a method for creating popup menus which appear on the screen without being entered in the System Menu. This is done with the function `nwCreateMenuWindow()`.

```
static void submenu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}
static NWS_MENUTABLE submenu_table[] = {
    {NWD_MF_NORMAL | NWD_MF_GRAYED, "Sub Menu 1", submenu_callback, 0},
    {NWD_MF_NULL, "", NULL, 0},
};
static void menu_callback(NWHWND hWndMenu, Sint32 idx, Sint32 param)
{
}
static NWS_MENUTABLE menu_table[] = {
    {NWD_MF_NORMAL, "Test Menu 1", menu_callback, 0},
    {NWD_MF_SEPARATOR, "", NULL, 0},
    {NWD_MF_POPUP, "Test Popup 1", NULL, (Sint32)submenu_table},
    {NWD_MF_NULL, "", NULL, 0},
};
void create_popup_menu(void)
{
    NWHWND hWnd;
    hWnd = nwCreateMenuWindow(menu_table, "Test Menu", 10, 10, NULL);
}
void njUserInit(void)
{
    :
    nwInitSystem(...);
    :
    create_popup_menu();
}
```

Here, the following popup menu is displayed on the screen.



Diagram 9-5. Popup Menu

The menu window we created will automatically be destroyed when a menu item is selected or the mouse is clicked outside the menu window area.

9.6.2 Creating a Popup Menu that Stays on the Screen

Because the menu window will automatically be destroyed when a menu item is selected or when the mouse is clicked outside the menu window area, the menu can only be selected once at most. It will be necessary to use `nwCreateMenuWindow()` and make the same menu.

The following code shows how to make a menu which stays on the screen

```
void create_popup_menu(void)
{
    NWHWND hWnd;
    hWnd = nwFindWindow(NULL, "Test Menu");
    if (!hWnd)
        hWnd = nwCreateMenuWindow(menu_table, "Test Menu", 10, 10, NULL);
}
Sint32 njUserMain(void)
{
    :
    :
    create_popup_menu();
    :
    :
    return nwExecute();
}
```

Every frame it checks to see if the menu window already exists and if it doesn't it recreates it. In this way, the popup menu appears to stay on the screen.

9.7 Description of Functions Used in Creating Popup Menus

<code>nwCreateMenuWindow</code>	Menu Function
Format	<code>NWHWND nwCreateMenuWindow(NWS_MENUTABLE *menuTbl, Sint8 *caption, Sint32 x, Sint32 y, NWHWND hWndParent);</code>
Parameters	menu- Array address of the menu table structure
Return value	If successful, it returns the window handle of the newly created menu window, else it returns NULL.
Function	Creates a popup menu window.
Reference	<code>nwDestroyWindow()</code> , <code>NWS_MENUTABLE</code> structure
Note	
Example	<code>NWHWND hWnd = nwCreateMenu(menu_tbl, "MENU", 100, 100, NULL);</code>

10. Mouse

10.1 Summary

Nindows does not have any special functions for the mouse. Getting the coordinates of the mouse cursor, button information is done with the Ninja functions.

10.2 Getting Mouse Information

Mouse information is acquired by using the Ninja function `njGetPeripheral()`. Please refer to the following example.

```
Sint32 njUserMain(void)
{
    Sint16 x, y;
    NJS_PERIPHERAL* mouse = njGetPeripheral(NJD_PORT_SYSMOUSE);
    x = mouse->x;           // X coordinates of the mouse cursor
    y = mouse->y;           // Y coordinates of the mouse cursor
    if (mouse->on & NJD_DGT_B0) {
        // The left button is being pressed
    }
    if (mouse->on & NJD_DGT_B1) {
        // The right button is being pressed
    }
    if (mouse->press & NJD_DGT_B0) {
        // The left button was pressed
    }
    if (mouse->press & NJD_DGT_B1) {
        // The right button was pressed
    }
    if (mouse->release & NJD_DGT_B0) {
        // The left button was released
    }
    if (mouse->release & NJD_DGT_B1) {
        // The right button was released
    }
    :
    :
    return nwExecute();
}
```

If you want to know what window is at the mouse cursor coordinates, do the following.

```
Sint32 njUserMain(void)
{
    Sint16 x, y;
    NWHWND hWnd;
    NJS_PERIPHERAL* mouse;
    static Sint16 old_x = 640 / 2, old_y = 480 / 2;
    mouse = njGetPeripheral(NJD_PORT_SYSMOUSE);
    hWnd = nwFindWindowByPos(old_x, old_y);
    if (hWnd) {
        // A window is displayed at the mouse cursor's coordinates
    } else {
        // A window is not displayed at the mouse cursor's coordinates
    }
    :
    old_x = mouse->x;
    old_y = mouse->y;
    return nwExecute();
}
```

10.3 Description of Functions Used for Getting Mouse Information

Here we will focus on Ninja peripheral functions and structures for the mouse.

Function	Description
njGetPeripheral	Gets information about peripherals

Nindows API

njGetPeripheral	Ninja Function
Format	NJS_PERIPHERAL* njGetPeripheral(long port)
Parameters	port- Peripheral port number Please specify NJD_PORT_SYSMOUSE to get information about the mouse
Return value	Address of the structure which stores the mouse information
Function	Gets information about the mouse.
Reference	NJS_PERIPHEAL structure
Note	This can be called many times per frame, but the information is changed as the frame is updated.
Example	<pre>Sint32 njUserMain(void) { NJS_PERIPHERAL* mouse = njGetPeripheral(NJD_PORT_SYSMOUSE); : }</pre>

Structure

NJS_PERIPHERAL Ninja Structure

Definition	<pre>typedef struct { Uint32 id; Uint32 on; Uint32 off; union { Uint32 push; Uint32 press; }; union { Uint32 pull; Uint32 release; }; Sint16 x; Sint16 y; Sint16 z; Sint16 r; Sint16 u; Sint16 v; Sint8* name; void* extend; Uint32 old; } NJS_PERIPHERAL;</pre>
Description	<p>This structure is not defined by Nindows, it is a Ninja structure. It stores information about joysticks, the keyboard, mouse and other input devices.</p>
Members	<p>id- Peripheral ID(NJD_DEV_SYSMOUSE) on- The bit corresponding to the pressed button is 1. off- The bit corresponding to the pressed button is 0. push, press- The bit corresponding to the button the moment it is pressed is 1. pull, release- The bit corresponding to the button the moment it is pressed is 0. x, y- The mouse coordinates are stored. z, r, u, v- Unused (reserved) name- Peripheral name extend- Unused (reserved) old- Reserved</p>
Reference	<p>njGetPeripheral()</p>

11. Fonts

11.1 Overview

Fonts can be changed only by selecting 'Font' from the Nindows System menu. This version supports only functions to acquire the typeface, width and height of a selected font.

11.2. Description of Font Functions

Function	Purpose
<code>nwGetFontSize</code>	Get the width and height of a font

Nindows API

<code>nwGetFontSize</code>	Font Function
Syntax	<code>Sint32 nwGetFontSize(Sint32* width, Sint32* height</code>
Parameters	<code>width, height</code> - Pointers to get font width and height
Return Value	Selected font typeface
Purpose	Get the typeface, width and height of the font selected on the System menu.
Reference	
Remarks	
Example	<pre>Sint32 width, height; nwGetFontSize(&width, &height);</pre>

11.3. Problems with Changing Fonts

Nindows does not automatically resize the window according to changes in font size. Also, parts of special windows and Properties controls may not display correctly with large font sizes.

