



Ninja Guide



Table of Contents

1 View Function	NGD-1
Use njInitView().	NGD-1
Directly set VIEW structure members.	NGD-2
View movement and rotation	NGD-3
Notes for using old View functions	NGD-3
The detail of the caution 1)	NGD-4
The detail of the caution 2)	NGD-4
The correct/incorrect examples using old View functions.	NGD-4
 2 Reminders of Ninja Motion	 NGD-7
Concept of motions in Ninja	NGD-7
Procedure to produce motions in Ninja	NGD-9
 3 How to Realize an Effective Strip	 NGD-11
The way of connecting vertices of a strip	NGD-14
Material and Texture	NGD-15
Comparison of expressions of a strip	NGD-15
Index type structure	NGD-15
Direct expression structure of vertices	NGD-16
Data reduction rate by strip	NGD-17

4 Basic Model Specification	NGD-19
Model Structures	NGD-20
Description of Structures	NGD-21
Model Structures	NGD-26
Meshsets	NGD-27
Texture Structures	NGD-31
Ninja Attributes	NGD-34
Texture Format	NGD-38
5 Motion Specification	NGD-41
Object Structure	NGD-41
Basic object structure	NGD-41
Chunk object structure	NGD-42
Explanation of evalflags	NGD-42
Camera Structure	NGD-43
Light Structure	NGD-43
Motion Structures	NGD-46
Explanation of Structure	NGD-47
Object Motion	NGD-51
Explanation of Structure	NGD-52
Camera Motion	NGD-53
Light Motion	NGD-54
Other Information	NGD-55
6 NINJA LIGHT	NGD-57
void njCreateLight(NJS_LIGHT*, Int)	NGD-57
void njDeleteLight(NJS_LIGHT*)	NGD-57
void njLightOff(NJS_LIGHT*)	NGD-57
void njLightOn(NJS_LIGHT*)	NGD-58
void njMultiLightMatrix(NJS_LIGHT*, NJS_MATRIX*)	NGD-58
void njSetLight(NJS_LIGHT*)	NGD-58
void njSetLightAlpha(NJS_LIGHT*, Float)	NGD-58
void njSetLightAngle(NJS_LIGHT*, NJS_Angle, NJS_Angle)	NGD-58
void njSetLightColor(NJS_LIGHT*, Float, Float, Float)	NGD-58
void njSetLightDirection(NJS_LIGHT*, Float, Float, Float)	NGD-59
void njSetLightIntensity(NJS_LIGHT*, Float, Float, Float)	NGD-59
void njSetLightLocation(NJS_LIGHT*, Float, Float, Float)	NGD-59
void njSetLightRange(NJS_LIGHT*, Float, Float)	NGD-59
void njSetUserLight(NJS_LIGHT*, NJF_LIGHT_FUNC*)	NGD-60
void njUnitLightMatrix(NJS_LIGHT*)	NGD-60
void njTranslateLightV(NJS_LIGHT*, NJS_VECTOR*)	NGD-60
void njTranslateLight(NJS_LIGHT*, Float, Float, Float)	NGD-60
void njRotateLightX(NJS_LIGHT*, NJS_Angle)	NGD-60

void njRotateLightXYZ(NJS_LIGHT*, NJS_Angle, NJS_Angle, NJS_Angle)	NGD-61
void njRotateLightY(NJS_LIGHT*, NJS_Angle)	NGD-61
void njRotateLightZ(NJS_LIGHT*, NJS_Angle)	NGD-61
Macro	NGD-61
How to use	NGD-62
LIGHTstructure Specification	NGD-65
The members of NJS_LIGHT structure	NGD-65
The members of NJS_LIGHT_ATTR structure	NGD-66
The members of NJS_LIGHT_CAL structure	NGD-68
 7 Scroll Guide	NGD-69
Ver.0.04	NGD-69
Ver.0.05	NGD-69
Image Units as Related to Scrolling	NGD-70
Overview	NGD-70
Image Units	NGD-70
Scroll Rotation, Resizing, and Movement	NGD-71
Overview	NGD-71
Scroll Rotation, Resizing, and Movement	NGD-71
Scroll Programming	NGD-72
Overview	NGD-72
Example of Programming a Scroll	NGD-72
Color	NGD-75
Overview	NGD-75
Color Mode	NGD-75
Scroll function, Structures, and Definitions	NGD-76
Overview	NGD-76
Scroll-related Functions	NGD-76
Scroll-related Structure	NGD-77
Scroll-related Definitions	NGD-77
Texture Structures for Use in Cell Programming	NGD-78

8 Texture Guide	NGD-79
Overview	NGD-79
Creating Textures	NGD-81
Overview	NGD-81
PVR Format	NGD-81
Category Code	NGD-82
Color Format	NGD-83
Memory	NGD-84
Overview	NGD-84
Texture Memory	NGD-84
Cache	NGD-84
Loading Textures	NGD-85
Overview	NGD-85
Flowchart of Texture Loading	NGD-85
Setting a Texture Buffer	NGD-86
Setting Cache Buffer	NGD-88
Creating a Texture List	NGD-89
Texture Numbers	NGD-91
Global Index Number	NGD-91
Automatic allocation of Global Index Number	NGD-92
Texture Load Error	NGD-92
Memory Texture	NGD-94
Render Texture	NGD-95
Texture functions, Structures, and Definitions	NGD-97
Overview	NGD-97
Texture Functions	NGD-97
Texture Structures	NGD-117
Texture Definitions	NGD-118
Sample Program	NGD-120
Overview	NGD-120
Sample	NGD-120
Notes for Texture functions	NGD-124
Overview	NGD-124
Notes for Switchover from SET2 to SET4/SET5	NGD-124
Notes for using texture functions in SET5	NGD-124

9 Chunk Model Specifications	NGD-125
Chunk Model Features	NGD-126
Model Structures	NGD-127
Structure Diagram	NGD-127
Chunk Specifications	NGD-129
Chunk Types	NGD-129
Chunk Structure	NGD-129
Chunk NULL	NGD-130
Chunk End	NGD-130
Chunk Bits	NGD-131
Chunk Tiny	NGD-136
Chunk Material	NGD-137
Chunk Vertex	NGD-143
Chunk Volume	NGD-158
Chunk Strip	NGD-162
ASCII Output Precautions	NGD-173
 10 Nindows Tutorial	 NGD-175
Special Features of Nindows	NGD-175
Creating a Simple Nindows Application	NGD-176
Integrating Nindows	NGD-176
Description of Functions used in Integrating Nindows	NGD-177
Using Nindows and Nindows Utilities	NGD-179
Using Nindows	NGD-179
Nindows Utilities	NGD-180
Changing Fonts	NGD-184
Windows	NGD-185
Summary	NGD-185
Creating a Window	NGD-185
Creating a Child Window	NGD-186
Window Related Parameters	NGD-186
Description of Window Support Functions	NGD-186
Samples and a Description of Window Support Functions	NGD-189
Scroll Windows	NGD-196
Summary	NGD-196
Creating a Scroll Window	NGD-196
Description of Functions Used to Create a Scroll Window	NGD-196
Edit Windows	NGD-199
Summary	NGD-199
Creating and Using an Edit Window	NGD-199
Description of Functions Used in Creating Edit Windows	NGD-200
Description of Functions Used in Nindows' Debug Window Utility	NGD-201
Scrollbar Controls	NGD-201
Summary	NGD-201

Creating Scrollbar Controls	NGD-201
Description of Functions Used in Creating Scrollbar Controls	NGD-202
Creating Scrollbar Controls that Use Low-level Scrollbar Functions	NGD-203
Description of Low-level Scrollbar Functions	NGD-204
Button Controls	NGD-207
Summary	NGD-207
Creating a Button Control	NGD-207
Button Validity and Invalidity	NGD-207
Description of Functions for Button Controls	NGD-208
Menus	NGD-209
Summary	NGD-209
Creating and Entering Menu Tables	NGD-209
Menu Callback Functions	NGD-210
Checkmarks	NGD-210
Description of Functions for Entering User Menus	NGD-211
Creating Popup Menus	NGD-213
Description of Functions Used in Creating Popup Menus	NGD-213
Mouse	NGD-214
Summary	NGD-214
Getting Mouse Information	NGD-214
Description of Functions Used for Getting Mouse Information	NGD-214
Fonts	NGD-216
Overview	NGD-216
Description of Font Functions	NGD-216
Problems with Changing Fonts	NGD-216



1. View Function

1 Initialization method

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

There are two methods of initialization.

1.1 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

1.2 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.

2 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.

3 Notes for using old View functions

```
njMultiViewMatrix
njRotateViewX
njRotateViewY
njRotateViewZ
njRotateViewXYZ
njTranslateView
njTranslateViewV
njUnitViewMatrix
```

The above eight functions are left to keep compatibility with old Ninja Libraries.

As there is a possibility that these functions will be deleted from the library in future, please do not use these functions for new programs. If you use these functions unavoidably, please keep the following cautions.

- 1) Executes the above functions only after the `njSetView()` function.
- 2) After executing the above functions, please be sure to execute `njClearMatrix()`.

If you do not follow these two cautions, the library might not work correctly.

3.1 The detail of the caution 1)

The old View functions operate members of the structure and NJS_MATRIX m directly.

But in the current View functions,

```
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
```

each member of the above list are operated. Then members and NJS_MATRIX m are renewed by the njSetView function. Consequently, if a View is operated by using old View functions before executing njSetView() function, the matrix is overwritten by the njSetView() function

3.2 The detail of the caution 2)

A View must be reflected in a matrix stack.

Though this function is incorporated in the njSetView() function, because of the above reason, a View can not be reflected in a matrix stack by using the njSetView() functions.

Therefore, in case of using old View functions, please do not forget to execute old Matrix functions and njClearMatrix().

3.3 The correct/incorrect examples using old View functions.

? The correct example using old View functions

```
NJS_VIEW _view_;

njInitView(&_view_);
njSetView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
njClearMatrix();
```

? The incorrect example using old View functions

```
NJS_VIEW _view_;

njInitView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
njSetView(&_view_);
njClearMatrix();
```

As the matrix of the View is overwritten by the `njSetView()` function, a View is left as it was initialized.

? The incorrect example using old View functions2

```
NJS_VIEW _view_;

njInitView(&_view_);
njSetView(&_view_);
njTranslateView(&_view_, 0.f, 0.f, 1000.f); // etc
```

As the result of the execution of the `njTranslateView()` function is not reflected in the matrix stack, a View is left as it was initialized.

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;
```


2. Reminders of Ninja Motion

1 Concept of motions in Ninja

Usually multiple motions are provided to one model. A model is actuated by assigning translation, rotation and scaling necessary for a motion to hierarchical tree model. In Ninja, motion is expressed by the difference from the base pose. In other words, motion consists of only information necessary for parts which should be actuated. A base pose should be determined carefully, since it can reduce the differential information when it contains parameters for stationary joints in each motion.

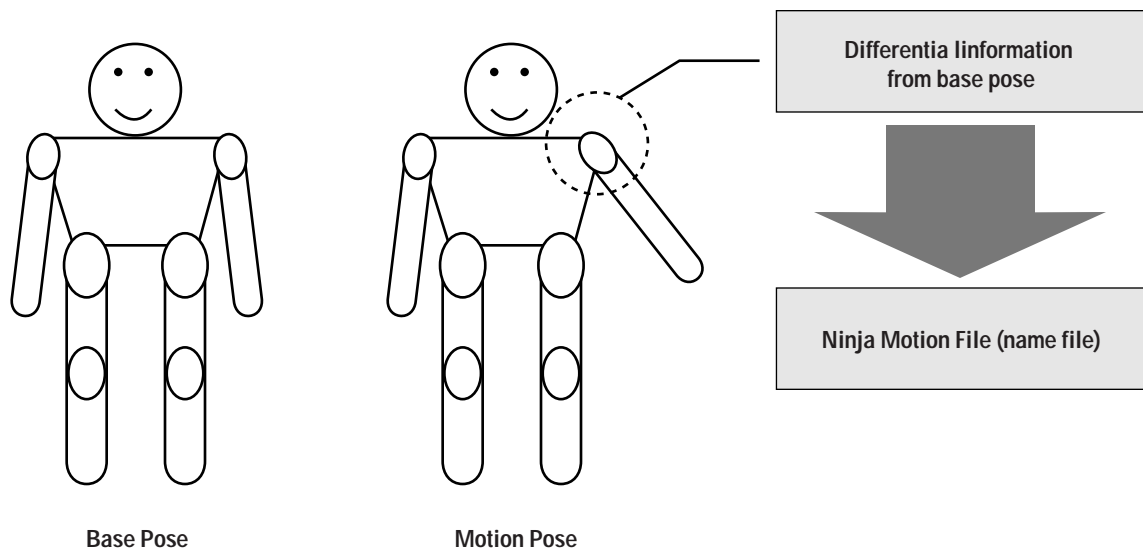


Figure 0.1 *Ninja Motion Concept*

There are three forms of motions in Ninja. To reduce the amount of data, they are selected according to the conditions.

Type A (TRRR)	Root node has translation and rotation, the other nodes have rotation only.
Type B (TRTR)	Each node has translation and rotation.
Type C (TRS)	Each node has translation, rotation and scaling.

Shape motion, interpolation methods (linear, spline, etc) are not described here. Please refer to other documentation for the format in detail.

In place of parameters skipped in type A and type B, translation, rotation, and scaling of each nodes of model tree data of base pose are used. Consequently, skipped data should be constant and coincide with the model tree of base pose. In some cases, rotation is skipped in type A. If the rotation of a node does not change during a motion and it remains as that in the base motion, the rotation is omitted.

By the method described thus far, Ninja retains motion data to small amount.

2 Procedure to produce motions in Ninja

- <step 1> Determine base pose.
- <step 2> Produce nja file and mrs file by converting a model. Here, mrs file implies motion resources, which is an information file containing two-level hierarchy and values for translation, rotation and scaling for each node.
- <step 3> Convert to nam file using mrs file containing base pose information. These motion data become differential information from single base pose.

Basically, a model and its motions are stored in one scene on a modeler, and can be converted simultaneously. When they are converted simultaneously, please note that multiple motions cannot be conducted using one base pose, since differences are produced based on the pose in the scene as a base pose, and the base pose changes at every motion. For a model requiring motions, make sure to produce a base pose model and mrs file which is a base for display first and to produce motions based on this model.



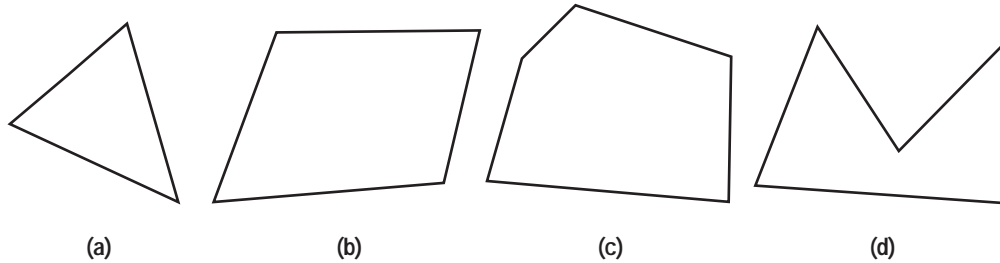
3. How to Realize an Effective Strip

Warning:

- This document is under preparation.
 - The current specifications and format for the strip is preliminary.
-

1 What is a strip?

A strip means a continuous polygon. Conventionally, a polygon means a surface consisting of three or more vertices.



Reentering polygon (d) is not used due to the specifications of the hardware.

Figure 1.1 Examples of Polygons

A Strip reduces the amount of data and calculation and increases the performance by treating neighboring multiple polygons as one data by connecting them (when applied to triangle polygons, it is called triangle strip). In addition, improving bus transfer neck by reducing the amount of transferred data to drawing hardware effectively enhances the peak performance (This assumes that the hardware has a function to process strip data. This hardware supports triangle strip).

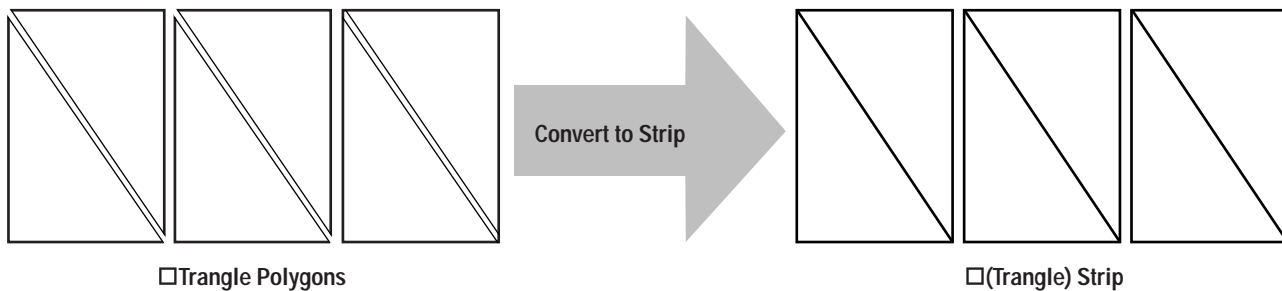


Figure 1.2 Example of Triangle Strip

The following section describes why the data is reduced compared with independent triangle polygons.

It can be considered that a quadrangle polygon consists of two triangle polygons. It can also be considered that a quadrangle is a strip produced by connecting two triangle polygons. Since three vertices are necessary for describing a triangle polygon, six vertices are required for two triangle polygons. At the same time, quadrangle polygon needs four vertices which are two vertices less.

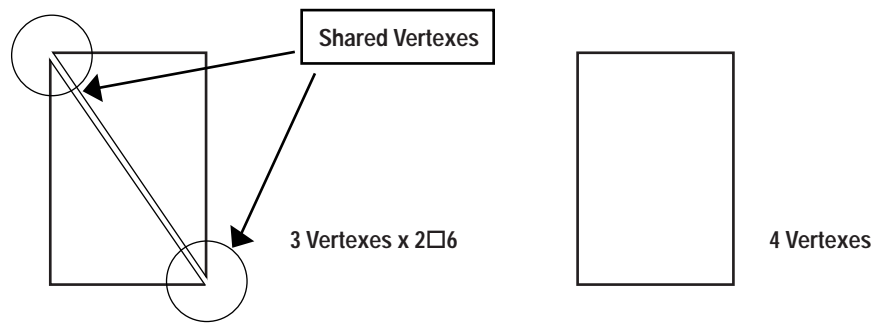


Figure 1.3 Comparison of Number of Vertices in Quadrangle Expression

This is because two vertices on each side of contacting lines of two triangles can be shared. A polygon model in a game consists of triangle polygons covering it in three dimension, and contains a lot of shared vertices. The idea of strip is to increase the expression efficiency by sharing vertices. The reduction rate of data is as follows.

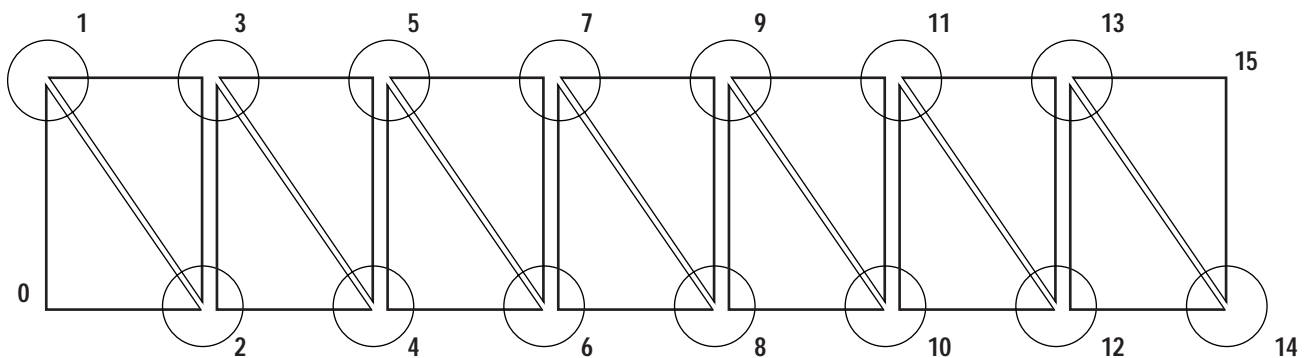


Figure 1.4 Shared Vertices in Strip

Except for vertex 0 and 15, other vertices are shared. In addition vertices 2 – 13 are shared by consecutive three triangle polygons. In other words, in a strip, the same result of color calculation can be used two or three times except for the first and the last vertices. As a result, the amount calculation can be reduced. From the second triangle onward, a triangle can be expressed by assigning one vertex in addition to two vertices from the previous triangle. The number of necessary vertices for a strip connecting N triangle polygons.

Number of vertices of a strip = $3 \text{ (Number of vertices of the first triangle)} + (N-1)$

In figure above, 14 triangle polygons are connected.

$$3 + (14 - 1) = 16 \text{ (16 from 0 to 15)}$$

The following is observed.

Except for the first triangle, one triangle can be expressed by one vertex in a strip. Consequently if a strip is long enough, the data necessary for a triangle polygon can be one vertex. When triangle polygons consisting a model are connected in a row and the row is long enough, three vertices for an independent triangle polygon can be reduced to one vertex and the amount of data is reduced to one third (33.333...%) of the original data amount. This gives the theoretical limit of data reduction of a triangle strip.

2 The way of connecting vertices of a strip

The way of expressing a polygon is determined by the way of listing vertices of a polygon. There are two expressions, clockwise and counterclockwise.

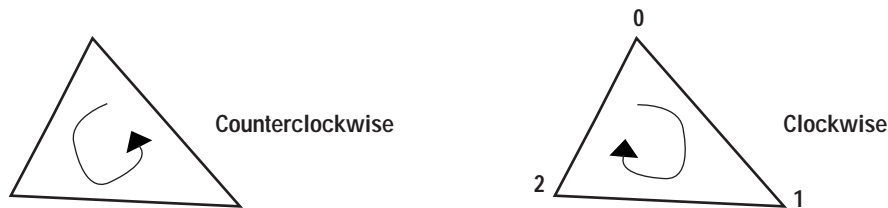


Figure 1.5 Polygons Expressed by Vertices

Choosing counterclockwise or clockwise is a matter of culture and either can do as a convention. In this documentation, it is assumed that usually an independent polygon is expressed by listing vertices counterclockwise. Also this direction of listing has a important role to determine the outer side of a polygon in 3D space. If the following vertices in ascending order give a counterclockwise rotation, the surface is defined to be the outer side. Otherwise, the surface is defined to be the inner side. Usually the inner side of a polygon cannot be observed. If it is concluded that the inner side of a polygon is observed from the viewpoint, the polygon becomes a subject of culling (deleting from drawing list).

In a strip, since it is necessary to use two vertices of the previous polygon, clockwise and counterclockwise appears alternately. In a strip, the processing is conducted assuming that clockwise and counterclockwise is the outer side alternately.

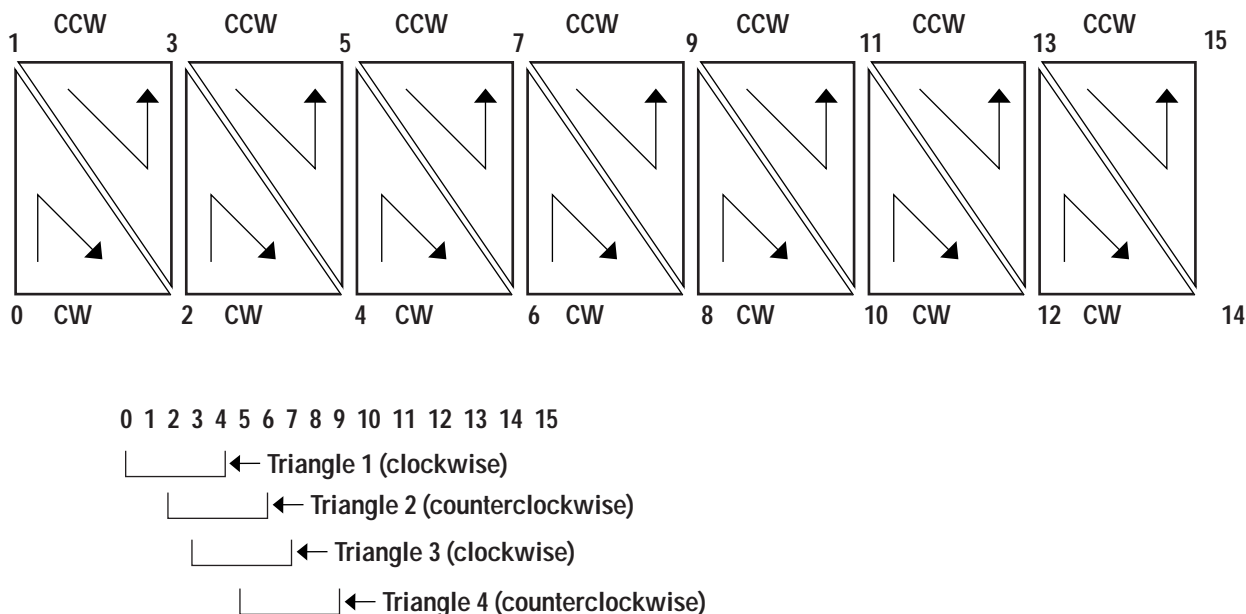


Figure 1.6 Clockwise and Counterclockwise Listing of Vertices in Strip

In Ninja, in order to make a strip as long as possible, both clockwise and counterclockwise are allowed for the direction of listing vertices of the first polygon. Clockwise is expressed by setting the flag at the MSB of the parameter for the length of a strip.

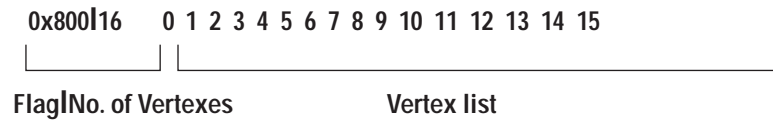


Figure 1.7 Example of Strip Structure in Ninja

3 Material and Texture

As described in Figure .4, a vertex of a strip is used three times by neighboring triangles. For that purpose, the information used by each triangle must be the same. A vertex has the information of the normal, the material, the texture and the UV value used for Gouraud calculation. Only neighboring triangles with vertices which have the same information and thus can be shared can be connected as a strip data.

4 Comparison of expressions of a strip

The following two forms of strip list (also applicable to polygon list) are possible.

- Index type structure of vertex list
- Direct expression structure of vertices

5 Index type structure

An index type consists of a vertex list and a strip list made up with the entry numbers of vertices.

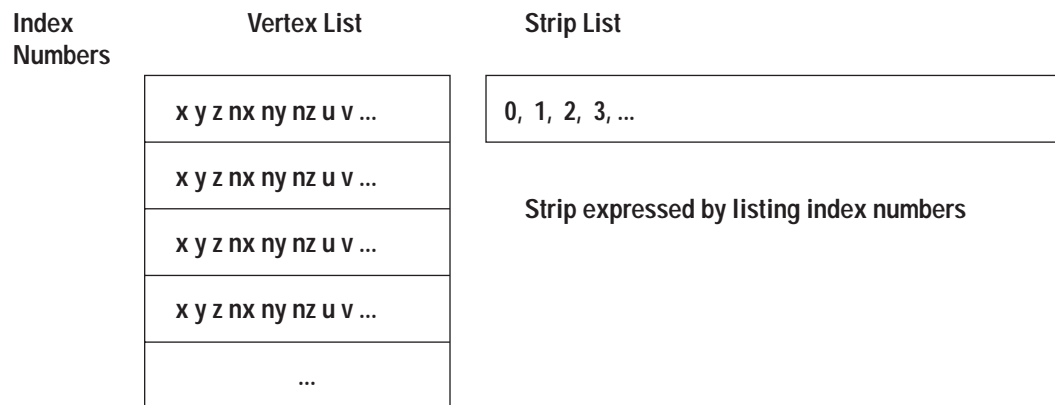


Figure 1.8 Index Type Strip

Merit

- Since vertices used in a model are arranged as a list without redundancy, a strip is expressed by referring results calculated before in stead of calculating again.
- Since data used more than once are expressed by indexes, the amount of data is reduced compared with the case where vertices are assigned directly.

Demerits

- Since an index and a vertex list are stored in different addresses, it tends to introduce CPU cache errors (this problem can be solved by a design suppressing cache errors).

In Ninja, index type strip is utilized.

5.1 Direct expression structure of vertices

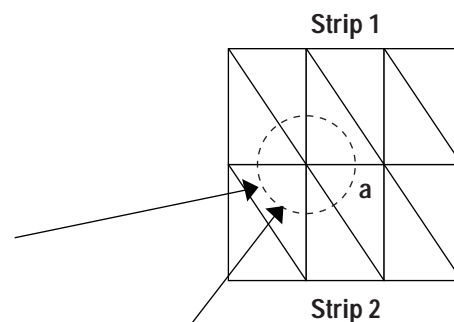
A strip is composed by listing vertex data directly.

Strip 1 List

x y z nx ny nz u v ...
x y z nx ny nz u v ...
x y z nx ny nz u v ...
x y z nx ny nz u v ...
...

Strip 2 List

x y z nx ny nz u v ...
x y z nx ny nz u v ...
x y z nx ny nz u v ...
x y z nx ny nz u v ...
...



Shared vertices are used both in Strip 1 and Strip 2.
Since in direct expression structure of vertices, vertex information is directly written into each strip list, vertex information data is doubled regarding to vertex a in size

Figure 1.9 Vertex direct expression strip

A strip list is composed by listing vertex data directly. If a vertex is used in multiple strips, the vertex is registered in a list each time when it is used. Compared with assigning a vertex by an index, the amount of data increases. It is supposed that the frequency of using one vertex is 1 - 10 (4 in average (estimation)) for an independent vertex, and is 1 - 3 (2-2.5 in average (estimation)) for a data model connected well by a strip. Therefore, the amount of data (in a simple direct expression) is larger than that in an index type strip. To compensate this large amount of data, such an approach as restraining the increase of the amount of data to some extent by reducing the number of bits expressing vertices and normals becomes necessary. Each time when a vertex is used, the 3D calculation should be conducted again. In case of a vertex used four times, four times of calculation will occur compared with the index type.

Merits

- Cache errors can be avoided since the data address is sequential.
- A buffer storing a calculated vertex list is not necessary.
- Since unit of processing can be each polygon, a small program can be executed at high speed.

Demerits

- Data can be large (a few times in simple estimation). However it can be suppressed to a certain degree by enhancing strip performance and reducing precision.
- It is difficult to express vertex animation.
- 3D calculations for the same vertex should be repeated as frequently as the vertex is used.

6 Data reduction rate by strip

The following section summarizes the expected amount of data reduction by the strip. The items listed below are reminders for that estimation.

- The strip (the strip means a triangle strip here) is a group of an independent triangle polygons which consist of originally sharable vertices.



4. Basic Model Specification

1 Overview

Besides the Basic Model format described in this document, Ninja also supports the Chunk Model format. While a drawing function is executed in the Chunk Model, the data are placed in a continuous memory space so as to maintain integrity of the SH4 cache. Expandability, flexibility, and data expression efficiency are excellent. In future, further tuning will be carried out, centering on the Chunk Model. The Basic Model is supported, but does not include the new features.

In the Chunk Model, the model structure contents have been significantly changed. The object structure is not changed, except for the fact that the model structure pointers have been altered to the Chunk Model.

Motions and textures besides the model use the same format as before. However, for compatibility with camera and light, the format of structure members has been changed.

For information on the Chunk Model, refer to the Chunk Model Specifications.

2 Model Structures

Object Tree

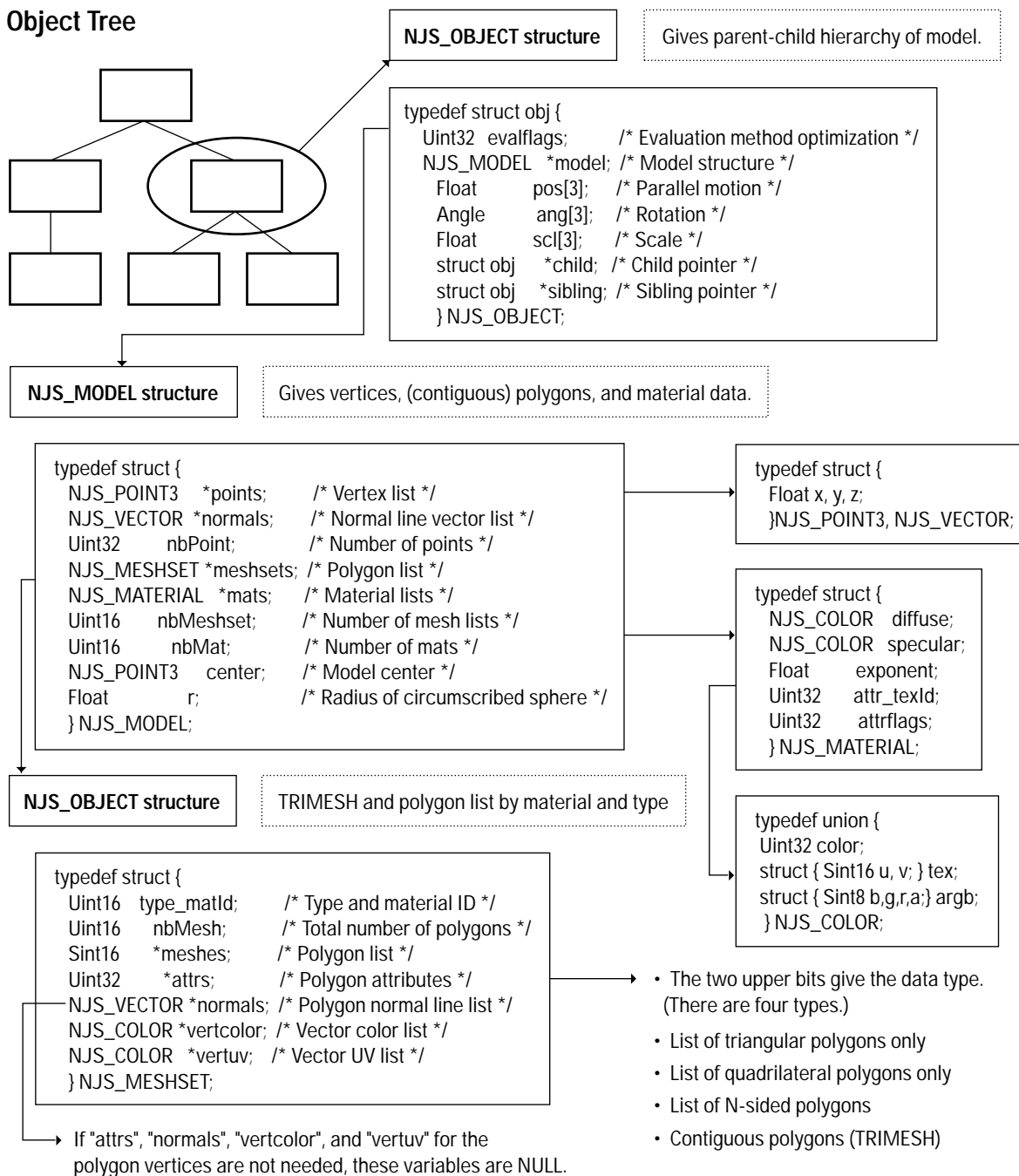


Figure 1.1 Diagram of Structures

2.1 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;
```

- Gives the parent/child structure of the model.
- Polygons and TRIMESHes (contiguous polygons) are set in "model".

Explanation of evalflags

```
#define NJD_EVAL_UNIT_POSBIT_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 expected by
                                /* LightWave3D
#define NJD_EVAL_SKIP BIT_6 /* Skip motion */
#define NJD_EVAL_SHAPE_SKIPBIT_7 /* Skip shape motion */
#define NJD_EVAL_MASK 0xff /* Mask for extracting above bits
```

These flags are set by the converter.

- `NJD_EVAL_UNIT_POS` is set when the parallel motion amount is "0". Parallel motion matrix processing is omitted when this flag is set.
- `NJD_EVAL_UNIT_ANG` is set when the rotation angle is "0". Rotation matrix processing is omitted when this flag is set.
- `NJD_EVAL_UNIT_SCL` is set when the scale is "1" for x, y, and z. Scale matrix processing 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 processing steps are omitted, and the matrix "push pop" operation is also omitted.
- The `NJD_EVAL_HIDE` 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` 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 "XYZ" in Ninja, the `NJD_EVAL_ZXY_ANG` is provided for execution via a library with the LightWave3D evaluation sequence. When this flag is set to ON, the rotation processing sequence is changed to "ZXY".
- The `NJD_EVAL_SKIP` indicates that this node does not include motion data. During motion execution, matrix processing is carried out using the object structure value without incrementing the motion node, and then proceeds to the next node. This allows motion also with polygon models having a different configuration, provided that the bone structure is the same.
- The `NJD_EVAL_SHAPE_SKIP` indicates that this node does not include shape motion data.

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_MEMORYBIT_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_PALETTE4(0x0500)
#define NJD_TEXFMT_PALETTE4_MM(0x0600)
#define NJD_TEXFMT_PALETTE8(0x0700)
#define NJD_TEXFMT_PALETTE8_MM(0x0800)
#define NJD_TEXFMT_RECTANGLE(0x0900)
#define NJD_TEXFMT_STRIDE (0x0B00)
#define NJD_TEXFMT_SMALLVQ(0x1000)
#define NJD_TEXFMT_SMALLVQ_MM(0x1100)
#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.

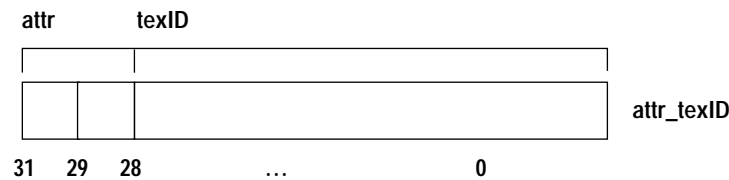
Texture list structure

```
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;
```



- "attr_texId" specifies a texture number in the current texture list "texlist". "attr" area is not currently used.
- 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 3.3, "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, "Meshsets" section in, "Construction of a Model" section.
For details on the attributes that are set in "attrs", refer to, "Ninja Attributes" section.

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; /* The center of the model */
    Float           r; /* Radius of circumscribed sphere */
                      /* from the center of the model */
} 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.

3 Model Structures

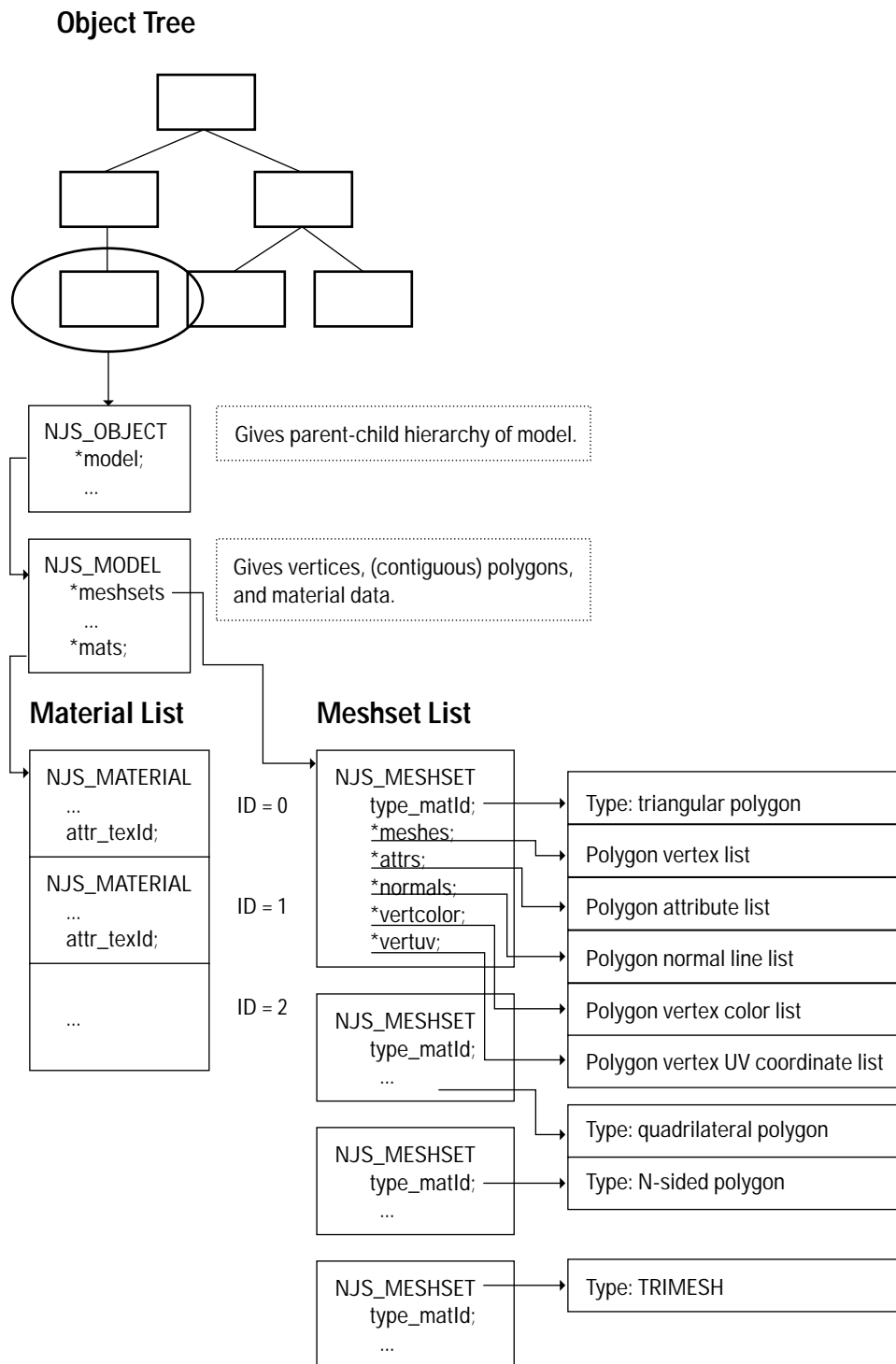
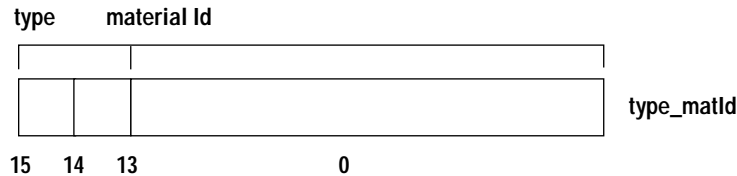


Figure 1.2 Diagram of Structure

3.1 Meshsets

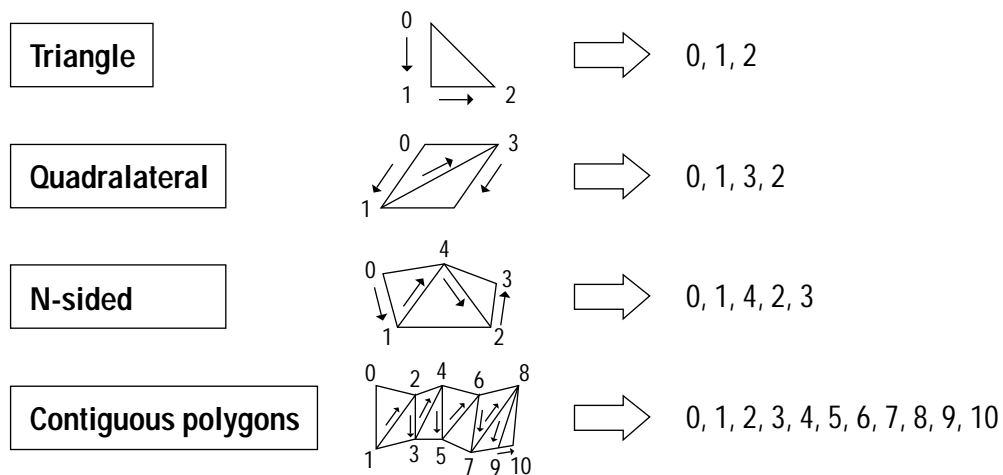
```
typedef struct {
    Uint16      type_matId; /* Type and material ID (0 to 16384)*/
    Uint16      nbMesh; /* Number of polygons/contiguous */
    /* polygons      */

```



```
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.



```
#define NJD_MESHSET_3 0x0000
#define NJD_MESHSET_4      0x4000
#define NJD_MESHSET_N      0x8000
#define NJD_MESHSET_TRIMESH 0xc000
```

An explanation of the data structure for each type follows.

For a Triangular Polygon List (NJD_MESHSET_3)

Example:

```

Polygon1 Polygon2

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

attrs[]           = NULL;

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

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

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

nbMesh            = number of vertices in "meshes"/3
```

- Attribute setting with "attrs" are not possible for individual polygons.
- 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 Quadrilateral Polygon List (NJD_MESHSET_4)**Example:**

```
Polygon1 Polygon2

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

attrs[]           = NULL;

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

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

nbMesh            = number of vertices in "meshes"/4
```

- Attribute setting with "attrs" are not possible for individual polygons.
- 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[]           = NULL;

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
```

- Attribute setting with "attrs" are not possible for individual polygons.
- 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 k'th trimesh).
- The NULL pointer is set for "attrs", "normals", "vertcolor", and "vertuv" if they are not needed.

Note: For high efficiency in joining trimesh shapes (triangle strips), Ninja supports start from trimesh right rotation and left rotation. When right rotation is used, the most significant bit of the start value indicating the length is set as a 1-bit flag.

Start from left rotation: length, vertex 1, vertex 2, ...

Start from right rotation: 0x800 | length, vertex 1, vertex 2, ...

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[] = NULL;
```

- Attribute setting with "attrs" are not possible for individual polygons.?

```
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.

3.2 Texture Structures

Object Tree

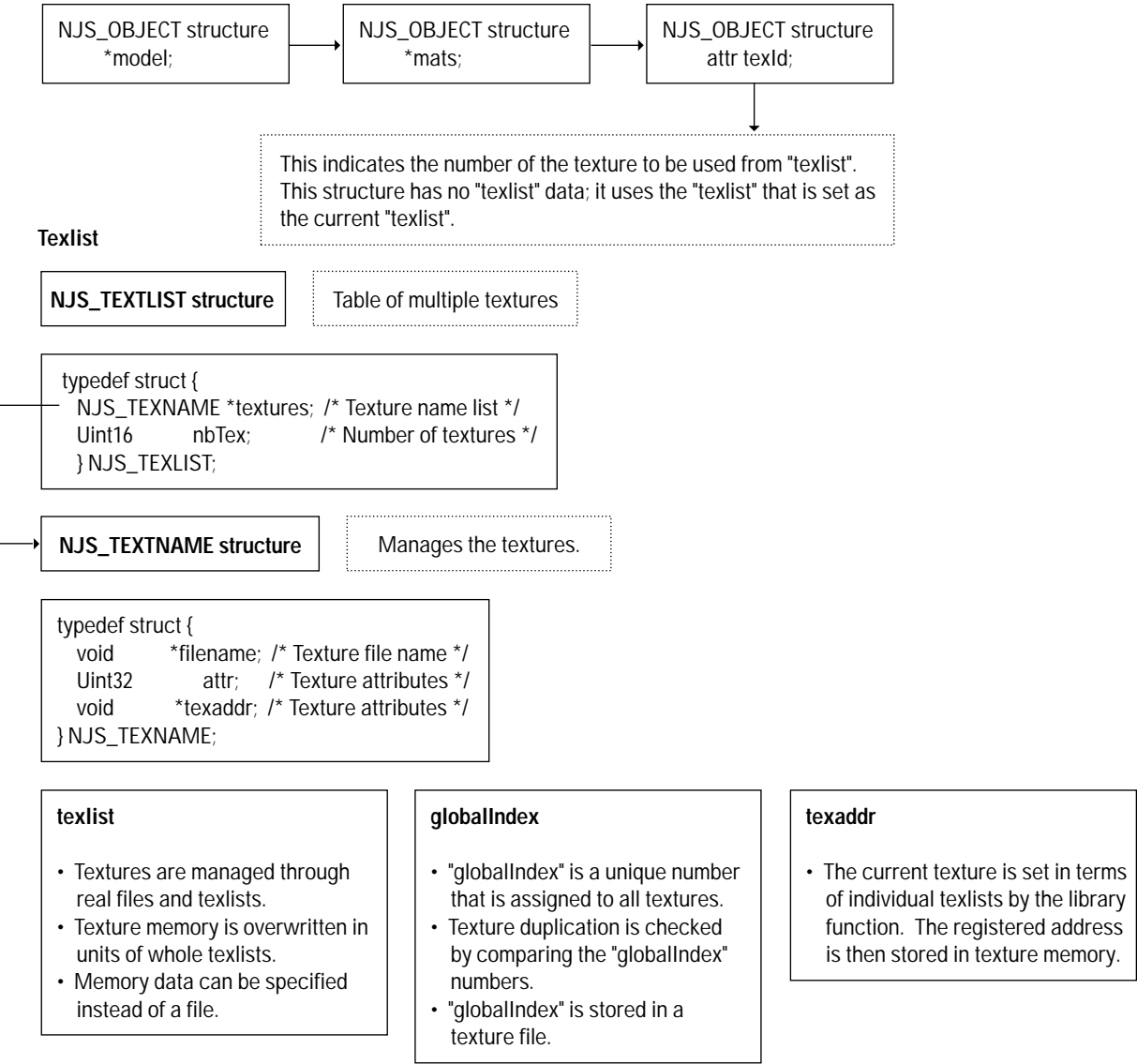


Figure 1.3 Diagram of Structure

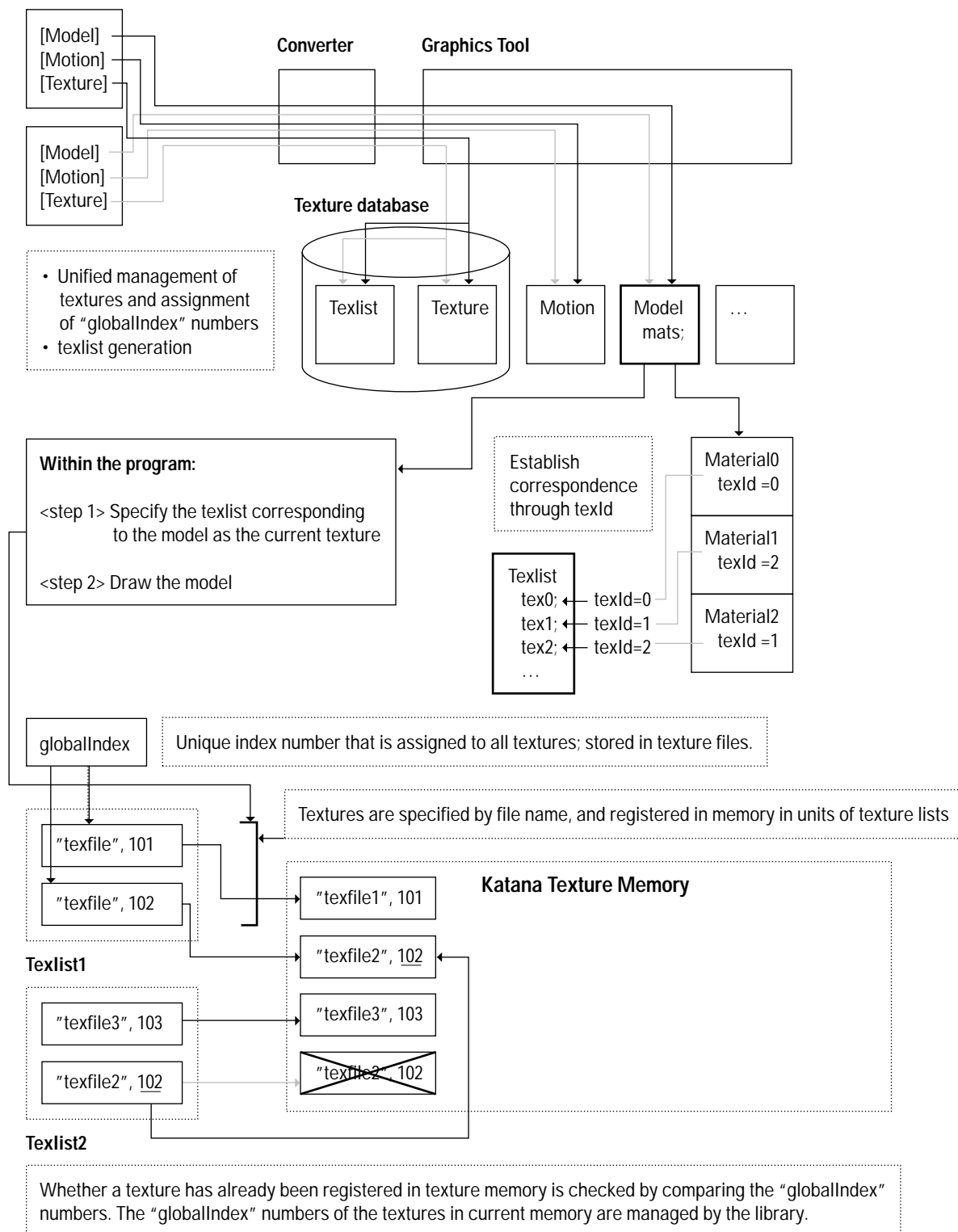
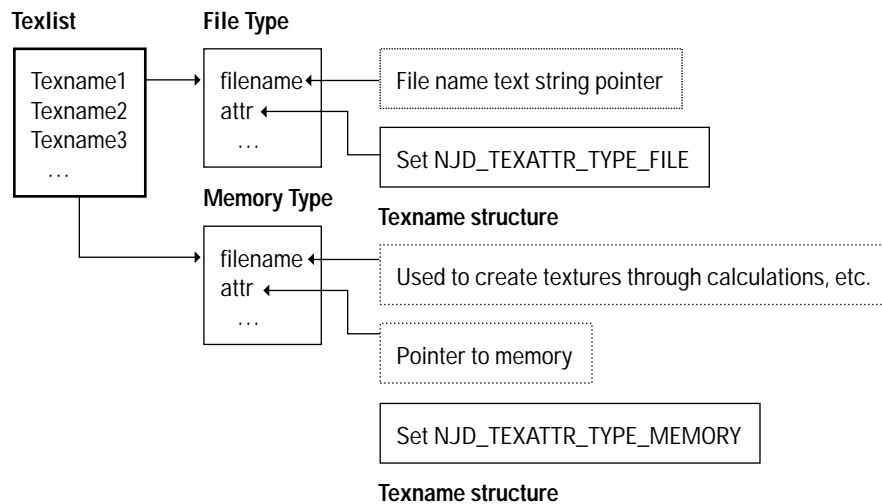


Figure 1.4 Overview of Texture Processing

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.

3.3 Ninja Attributes

The attribute defined here is used for "attrflags" of NJS_MATERIAL. The polygon-unit attribute "attrs" is always NULL. The polygon-unit attribute is used in the Chunk Model.

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

31-29:	SRC Alpha Instruction	?Alpha blending parameter; explained later?
28-26:	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"??
20:	Use Alpha	?Alpha enabled/disabled; enabled when "1"?
19:	Ignore Specular	?Ignores specular; disabled when "1"?
18-17:	Flip UV	?Flip control?
16-15:	Clamp UV	?Clamp control?
14-13:	Filter-Mode	0 ... Point Sampled(hard spec) 1 ... Bilinear Filter(hard spec) 2 ... Tri-liner Filter(hard spec)
12:	Super-Sample Texture	?Anisotropic Fliter ON/OFF?
11-8:	Mip-Map 'D' adjust	?16-step mip-map adjustment; explained later?
7:	Pick Status	?Stroes the status that has been picked?
6-0:	User Flags	?Not used?

- The bit string 25-21, 19, 14-13, 7-0 has a separate meaning for the hardware. This part is masked by the library, and the control value expected by the hardware is set and used by the library.
- 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 3-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' Colour	2	(OR, OG,OB,OA)
Inverse 'Other' Colour	3	(1 - OR,1- OG,1 - OB,1 - OA)
SRC Alpha	4	(SA, SA, SA, SA)
Inverse SRC Alpha	5	(1- SA, 1 - SA, 1 - SA, 1 - SA)
DST Alpha	6	(DA, DA, DA, DA)
Inverse DST Alpha	7	(1- DA, 1 - DA, 1- DA, 1 - DA)

"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	Reserved

- 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	Illegal
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_MASK      (BIT_31|BIT_30|BIT_29) /* MASK*/

/* DST Alpha Instr(31-29) */
#define NJD_DA_ZERO      (0)              /* 0 zero*/
#define NJD_DA_ONE      (BIT_26)         /* 1 one*/
#define NJD_DA_OTHER     (BIT_27)         /* 2 Other Color*/
#define NJD_DA_INV_OTHER (BIT_27|BIT_26)  /* 3 Inverse Other Color*/
#define NJD_DA_SRC       (BIT_28)         /* 4 SRC Alpha*/
#define NJD_DA_INV_SRC   (BIT_28|BIT_26)   /* 5 Inverse SRC Alpha*/
#define NJD_DA_DST       (BIT_28|BIT_27)   /* 6 DST Alpha*/
#define NJD_DA_INV_DST   (BIT_28|BIT_27|BIT_26) /* 7 Inverse DST Alpha*/
#define NJD_DA_MASK      (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 'D' 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)
#define NJD_FLAG_PICK (BIT_7)

/* 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)

```

3.4 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.

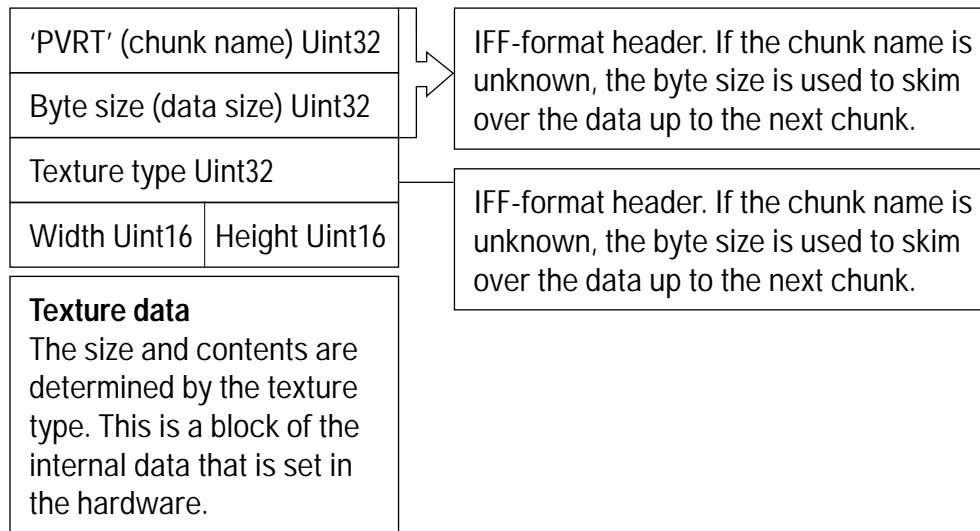
VQ

Performs texture compression using vector quantization. For details, refer to the VQ Specifications. Specifications for YUV422 and palette texture are pending.

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_SMALLVQ (0x1000)
#define NJD_TEXFMT_SMALLVQ_MM(0x1100)
#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 file stores the texture control data. The details are currently under study.



5. *Motion Specification*

1 Overview

Ninja defines model, camera, and light motion in a single structure.

Data arrays are used as keyframe setting units, with pointer tables defining the entire motion. This method uses motion only in the required sections, and keyframe interpolation can be performed for each parameter. Also, it is possible to reuse common motion parts for camera and light.

2 Object Structure

The object structure can be linked to other objects with child and sibling pointers, creating a parent/child hierarchic model. Motion in the hierarchic model is implemented by tracing the layers in the order child/sibling and combining the sorted motion data for the nodes in this sequence with the "pos", "ang", and "scl" values of the object structure. The "evalflags" serve for suppressing matrix processing and controlling other motion options. For information on the model data structure, refer to the Basic Model Specifications and the Chunk Model Specifications.

2.1 Basic object structure

```
typedef struct obj {  
    Uint32                evalflags;        /* Evaluation method optimization*/  
    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;
```

2.2 Chunk object structure

```
typedef struct cnkobj {
    Uint32          evalflags;      /* Evaluation method optimization*/
    NJS_CNK_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_CNK_OBJECT;
```

- Gives the parent/child structure of the model.
- Polygons and TRIMESHes (continuous polygons) are set in "model".

2.3 Explanation of evalflags

```
#define NJD_EVAL_UNIT_POSBIT_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 expected by LightWave3D */
#define NJD_EVAL_SKIPBIT_6
    /* Skip motion */
#define NJD_EVAL_SHAPE_SKIPBIT_7
    /* Skip shape motion */
#define NJD_EVAL_MASK0xff
    /* Mask for extracting above bits */
```

These flags are set by the converter.

- NJD_EVAL_UNIT_POS is set when the parallel motion amount is "0". Parallel motion matrix processing is omitted when this flag is set.
- NJD_EVAL_UNIT_ANG is set when the rotation angle is "0". Rotation matrix processing is omitted when this flag is set.
- NJD_EVAL_UNIT_SCL is set when the scale is "1" for x, y, and z. Scale matrix processing 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 processing steps are omitted, and the matrix "push pop" operation is also omitted.
- The NJD_EVAL_HIDE 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 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 "XYZ" in Ninja, the NJD_EVAL_ZXY_ANG is provided for execution via a library with the LightWave3D evaluation sequence. When this flag is set to ON, the rotation processing sequence is changed to "ZXY".
- The NJD_EVAL_SKIP indicates that this node does not include motion data. During motion execution, matrix processing is carried out using the object structure value without incrementing the motion node, and then proceeds to the next node. This allows motion also with polygon models having a different configuration, provided that the bone structure is the same.
- The NJD_EVAL_SHAPE_SKIP indicates that this node does not include shape motion data.

3 Camera Structure

The camera structure is as described below.

The motion parameters are position, vector, roll, and angle.

```
NJS_CAMERA structure
typedef struct{
    Float px, py, pz;           (Camera position)
    Float vx, vy, vz;           (Camera vector in unit direction [Local Z axis])
    Angle roll;                 (Camera roll)
    Angle ang;                  (Camera angle)
    Float n_clip;               (Near clip)
    Float f_clip;               (Far clip)
    NJS_VECTOR ??????           (Camera local X, Y axis)
} NJS_CAMERA
```

4 Light Structure

The light structure is as described below.

The motion parameters are position, vector, and color. For the spotlight, additional motion parameters are near limit value, far limit value, inner limit angle, and outer limit angle. For a detailed explanation of the light structure, refer to the section "Light Settings".

NJS_LIGHT structure

```
struct {
    NJS_MATRIX          mtrx;           ?Light source matrix?
    NJS_POINT3           pnt;           ?Light source position?
    NJS_VECTOR           vctr;           ?Light source vector in unit direction?
    BOOL                 stat;           ?Status: light source used/not used?
    Int                  reserve;        ?Reserved)
    NJS_LIGHT_CAL         ltcal;         ?Light calculation structure?
    NJS_LIGHT_ATTR        attr;          ?Attribute structure?
} NJS_LIGHT;
```

```
<stat>
#define NJD_LIGHT_ON      Reflects light
#define NJD_LIGHT_OFF     Does not reflect light
```

NJS_LIGHT_ATTR structure

```

struct {
    Int                lsrc;                ?Light source type?
    Float              ispc;                ?Specular light intensity: 0 to 1?
    Float              idif;                ?Diffuse intensity: 0 to 1?
    Float              iamb;                ?Ambient intensity: 0 to 1?
    Float              nrang;                ?Distance for maximum light intensity:
near limit value?
    Float              frang;                ?Distance for light intensity cutoff:
far limit value?
    void*              func;                ?Callback function pointer?
    Angle              iang;                ?Angle for maximum light intensity:
inner limit angle?
    Angle              oang;                ?Angle for light intensity cutoff:
outer limit angle?
    NJS_ARGB           argb;                ?Light color?
} NJS_LIGHT_ATTR
<lsrc>
Light source type

```

```

#define NJD_SPOT_LIGHT    Spotlight
#define NJD_DIR_LIGHT     Parallel light source
#define NJD_POINT_LIGHT   Point light
#define NJD_AMBIENT       Ambient light
#define NJD_SPEC_DIR      Parallel beam highlight
#define NJD_SPEC_POINT    Point beam highlight
#define NJD_LAMBERT_DIR   Parallel beam Lambert
#define NJD_LAMBERT_POINT Point beam Lambert
#define NJD_PHONG_DIR     Parallel beam Phong
#define NJD_PHONG_POINT   Point beam Phong
#define NJD_USER_LIGHT    User-defined light
#define NJD_BLOCK_LIGHT   Block light
    .
    .
    .

```

<ispc, idif, iamb>

Light balance (as per equation below)

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

Upper limit (lower lamp) is clamped.

<near, far>

Effective light range (distance) as specified by NJD_POINT_LIGHT (point light source), NJD_SPOT_LIGHT (spot light source), etc.

nrang Distance limit value where light is at upper limit value. Default: 1.f

frang Distance limit value where light processing is performed. Default: 65535.f

<iang, oang>
Effective light range (distance) as specified by NJD_SPOT_LIGHT (spot light source), etc.

iang Angle limit value where light is at upper limit value. Default: (DEG)10.f
oang Angle limit value where light processing is performed. Default: (DEG)30.f

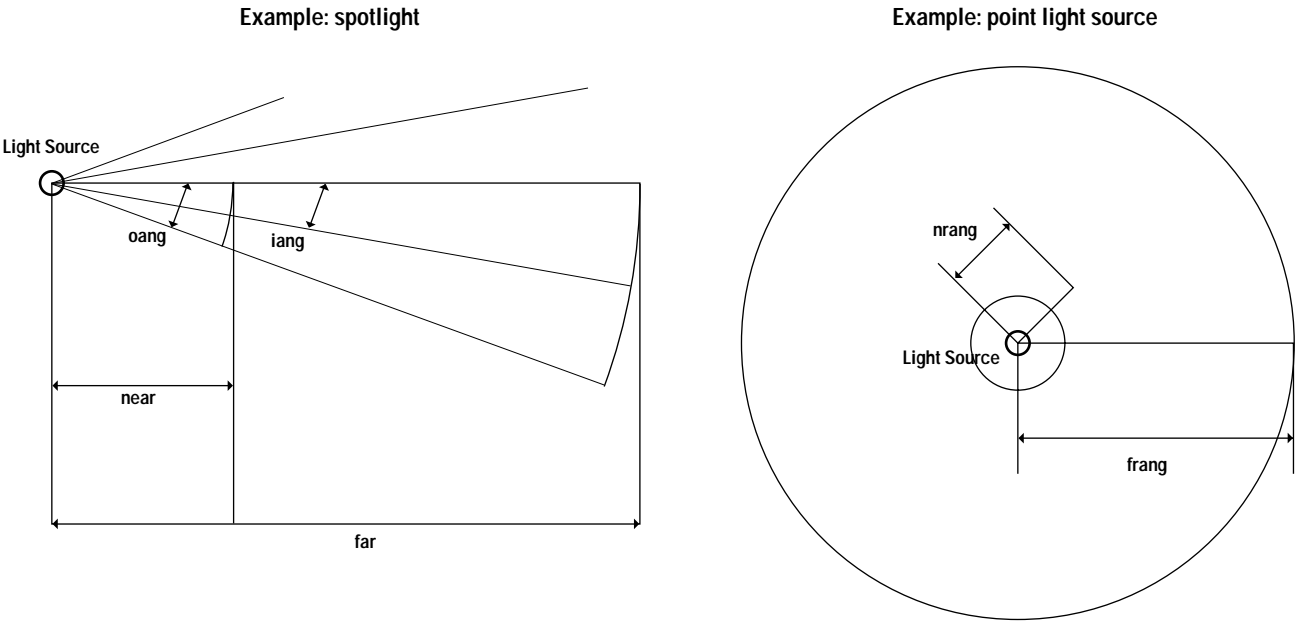
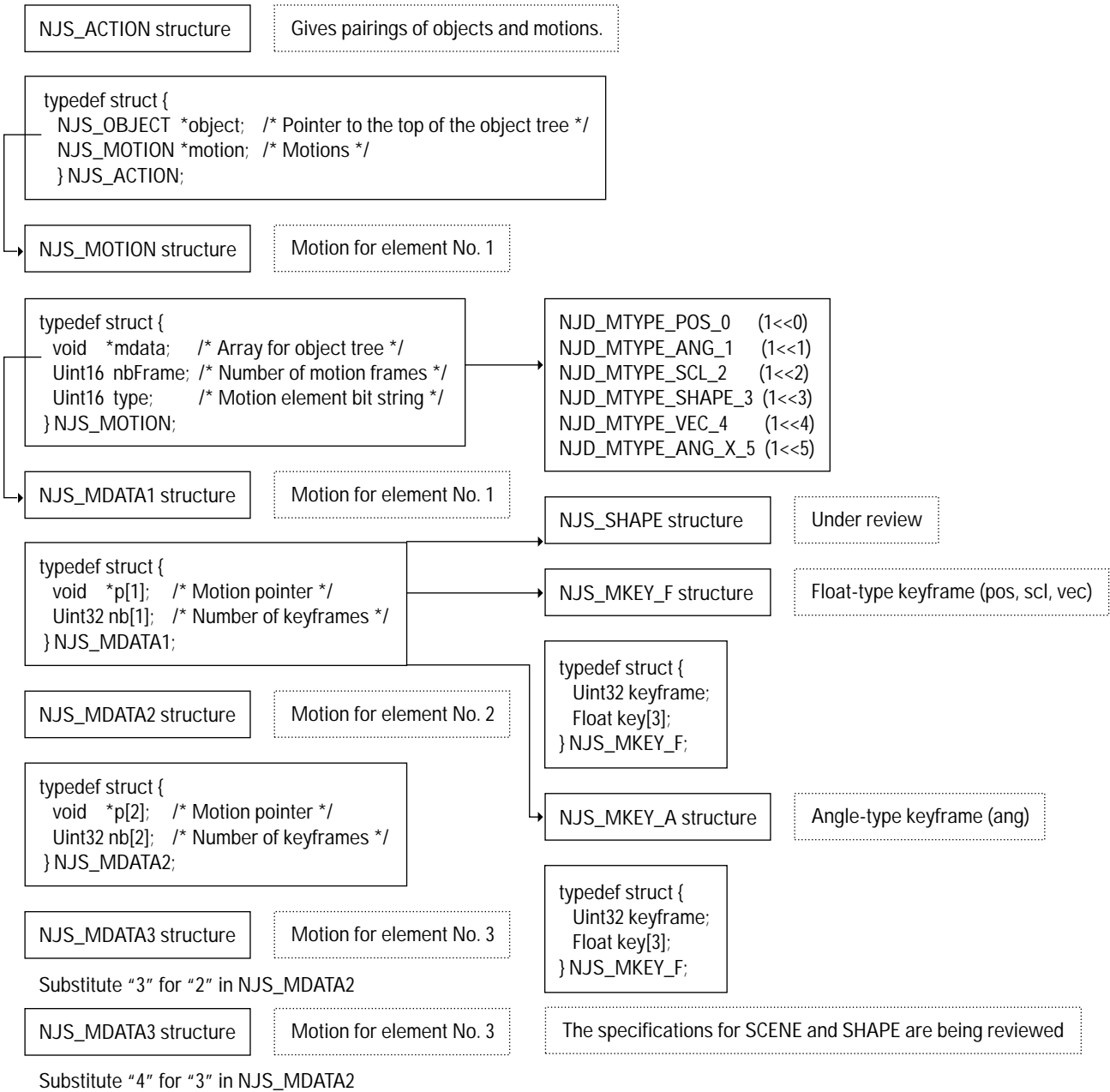


Figure 1.1 *Diagram of Structure*

5 Motion Structures



5.1 Explanation of Structure

Action structure for model

```
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".

Action structure for camera

```
typedef struct {
    NJS_CAMERA      *camera; /* Pointer to camera structure*/
    NJS_MOTION      *motion; /* Motion list*/
} NJS_CACTION;
```

Action structure for light

```
typedef struct {
    NJS_LIGHT       *light; /* Pointer to light structure*/
    NJS_MOTION      *motion; /* Motion list*/
} NJS_LACTION;
```

Motion structure

```
typedef struct {
    void            *mdata; /* Array for object tree*/
    Uint32          nbFrame; /* Number of motion frames*/
    Uint16          type; /* Motion element bit string*/
    Uint16          inp_fn; /* Interpolation method and number of elements*/
} NJS_MOTION;
```

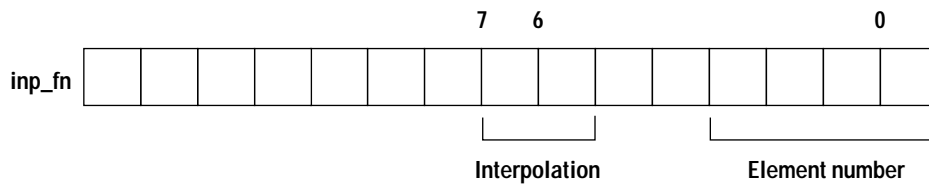
- "mdata" contains, in the form of an array, a number of NJS_MDATA sufficient for all of the NJS_OBJECTs included in the object tree.
- For NJS_MDATA, the NJS_MDATA1 to 5 structures are used according to the number of motion configuration elements.

```
#define NJD_MTYPE_POS_0      (1<<0) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_ANG_1     (1<<1) /* Uses NJS_MKEY_A*/
#define NJD_MTYPE_SCL_2     (1<<2) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_VEC_3     (1<<3) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_VERT_4    (1<<4) /* Uses NJS_MKEY_P*/
#define NJD_MTYPE_NORM_5    (1<<5) /* Uses NJS_MKEY_P*/
#define NJD_MTYPE_TARGET_3 (1<<6) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_ROLL_6    (1<<7) /* Uses NJS_MKEY_A1*/
#define NJD_MTYPE_ANGLE_7   (1<<8) /* Uses NJS_MKEY_A1*/
#define NJD_MTYPE_RGB_8     (1<<9) /* Uses NJS_MKEY_UI32*/
#define NJD_MTYPE_INTENSITY_9 (1<<10) /* Uses NJS_MKEY_F1*/
#define NJD_MTYPE_SPOT_10   (1<<11) /* Uses NJS_MKEY_SPOT*/
#define NJD_MTYPE_POINT_10 (1<<12) /* Uses NJS_MKEY_F2*/
```

- When normal motion includes the three elements of parallel motion ("pos"), rotation ("ang") and scale ("scl"), NJS_MDATA3 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    0x0000/* Linear interpolation*/
#define NJD_MTYPE_SPLINE  0x0040/* Spline interpolation*/
#define NJD_MTYPE_USER     0x0080/* User function interpolation*/
#define NJD_MTYPE_MASK     0x00c0/* 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 5 structure

```
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;
```

- All data are expressed as keyframes.
- Number of motion keyframes of `p[i]` element is inserted in `nb[i]`.

MDATA4 and MDATA5 are defined for the light source. MDATA5 is used only for spotlight light sources.

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

typedef struct {
    void                *p[5];           /* Motion pointer*/
    Uint32              nb[5];           /* Number of keyframes*/
} NJS_MDATA5;

Key structure
typedef struct {
    Uint32              keyframe;         /* Keyframe number*/
    Float               key[3];           /* Float type key value (array 3)*/
} NJS_MKEY_F;
```

- Used for parallel motion (POS), scale (SCL), and vector (VEC).

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

- Used for rotation (ANG).

```
typedef struct {
    Uint32              keyframe;         /* Keyframe number*/
    Void                *key;            /* Pointer*/
} NJS_MKEY_P;
```

- Used for shape (SHAPE).

```
typedef struct {
    Uint32              keyframe;         /* Keyframe number */
    Uint32              key;              /* Unsigned int32 type key value */
} NJS_MKEY_UI32;
```

- Used for light color.

```
typedef struct {
    Uint 32             keyframe;         /* Keyframe number */
    Sint32              key;              /* Signed int32 type key value */
} NJS_MKEY_A1;
```

- Used for camera roll (ROLL) and angle (ANGLE).

```
typedef struct {
    Uint32              keyframe;         /* Keyframe number */
    Float               key;              /* Float type key value */
} NJS_MKEY_F1;
```

- Used for light intensity (INTENSITY) and angle (ANGLE).

```
typedef struct {
    Uint32          keyframe;          /* Keyframe number          */
    Float           key[2];             /* Float type key value (array 3) */
} NJS_MKEY_F2;
```

- Used for point light source (POINT).

```
typedef struct {
    Uint32          keyframe; /* Keyframe number          */
    Float           near; /* Float type / near limit value key value */
    Float           far; /* Float type / far limit value key value */
    Angle           iang; /* Angle type / inner limit angle key value */
    Angle           oang; /* Angle type / inner limit angle key value */
} NJS_MKEY_SPOT;
```

- Used for spotlight (SPOT).

6 Object Motion

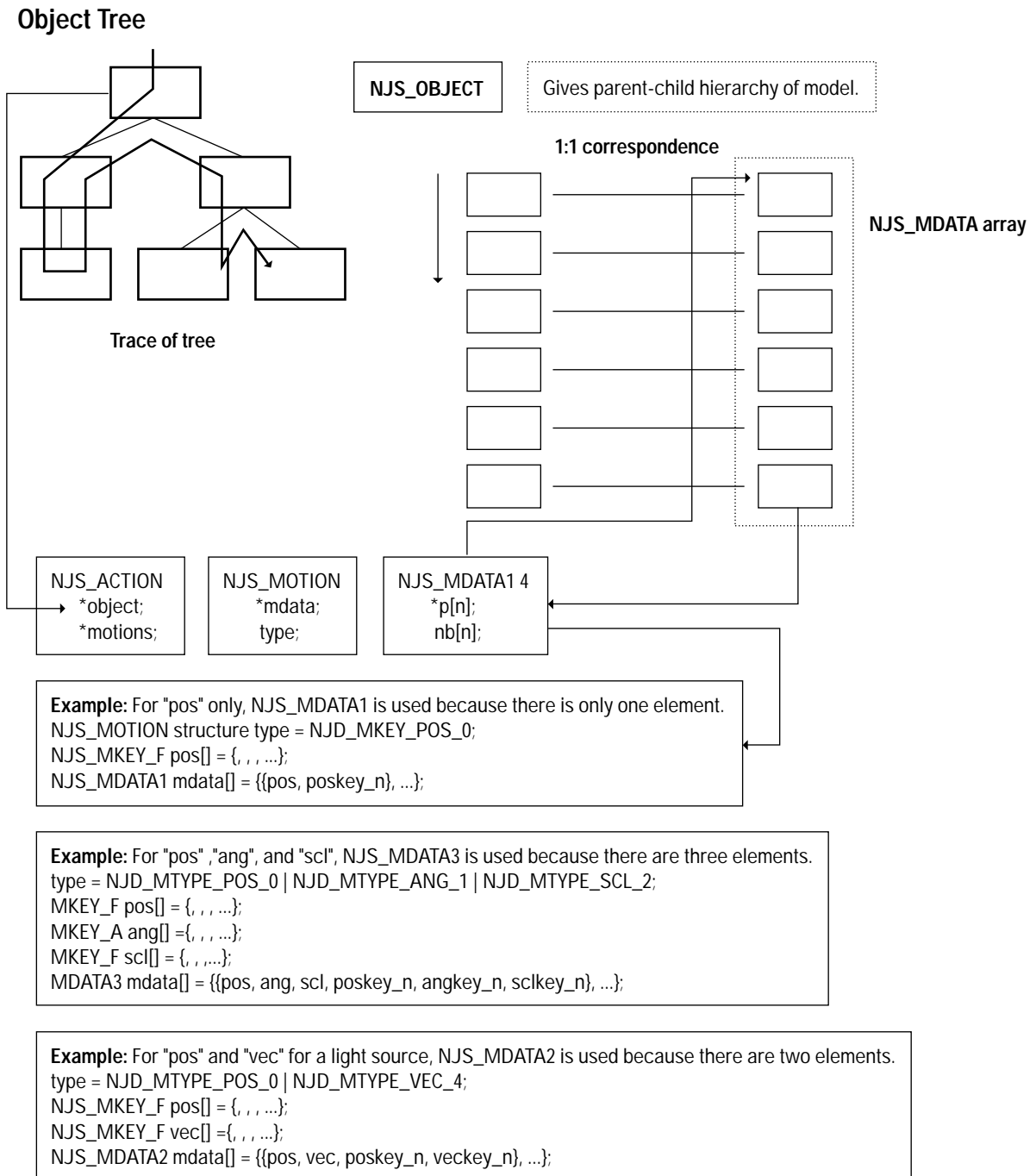


Figure 1.2 Diagram of Structure

6.1 Explanation of Structure

- 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 (Currently unsupported).
- The keyframe numbers start from zero. The negative value cannot be used.

Position	Parallel motion
Angle	Rotation
Scale	Enlargement/reduction
Vertex	Animation by polygon vertex motion (shape)
Normal	Normal line for animation by polygon verte

- Object motion has the above five elements.
- Because of problems that can occur when implementing the library, the shape data "Vertex" and "Normal" are output as separate data (.nas) from the "Position", "Angle", and "Scale" data (.nam). The maximum number of elements therefore is three. The structures NJS_MDATA1 through NJS_MDATA3 are provided for object motion. NJS_MDATA4 and NJS_MDATA5 are defined for light source and camera.
- 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_VEC_3    (1<<3) /* Use NJS_MKEY_F*/  
#define NJD_MTYPE_VERT_4    (1<<4) /* Use NJS_MKEY_P*/  
#define NJD_MTYPE_NORM_5    (1<<5) /* Use NJS_MKEY_P*/
```

- 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 upper 2 bits of "type".

```
#define NJD_MTYPE_LINER            0x0000  
#define NJD_MTYPE_SPLINE    0x0040  
#define NJD_MTYPE_USER            0x0080  
#define NJD_MTYPE_MASK            0x00c0
```

- `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".

7 Camera Motion

Because camera does not use a parent/child hierarchic configuration, it is basically the same as a motion structure for a single object.

The action structure uses `NJS_CACTION`.

Camera has the following four elements:

- Position (POS)
- Vector (VEC) or target (TARGET)
- Roll (ROLL)
- Angle (ANGLE)

These use `NJS_MTYPE_1` through `NJS_MTYPE_4`, as required.

```
#define NJD_MTYPE_POS_0 (1<<0) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_VEC_3 (1<<3) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_TARGET_3 (1<<6) /* Uses NJS_MKEY_F*/
#define NJD_MTYPE_ROLL_6 (1<<7) /* Uses NJS_MKEY_A1*/
#define NJD_MTYPE_ANGLE_7 (1<<8) /* Uses NJS_MKEY_A1*/
```

"Vec" stands for vector, and "Target" for the target position.

The common "3" in `NJD_MTYPE_VEC_3` and `NJD_MTYPE_TARGET_3` means that these cannot be used simultaneously.

- Free camera (with direction as vector)
`type=NJD_MTYPE_POS_0 | NJD_MTYPE_VEC_3 | NJD_MTYPE_ROLL_6 | NJD_MTYPE_ANGLE_7;`
- Target camera (with direction as target position and image angle animation)
`type=NJD_MTYPE_POS_0 | NJD_MTYPE_TARGET_3 | NJD_MTYPE_ROLL_6 | NJD_MTYPE_ANGLE_7;`
- Interpolation method is specified with the upper 2 bits of "inp_fn". This is the same as for object motion.

8 Light Motion

Because light does not use a parent/child hierarchic configuration, it is basically the same as a motion structure for a single object.

The action structure uses `NJS_LACTION`.

Light motion objects are either point light source, parallel light source, or spotlight.

Point light source has the following four elements:

- Position (POS)
- Range (POINT)
- Color (RGB)
- Intensity (INTENSITY)

The range element is comprised in the near limit value (NearRange) and far limit value (FarRange).

Parallel light source has the following four elements:

- Position (POS)
- Vector (VEC) or target (TARGET)
- Color (RGB)
- Intensity (INTENSITY)

For the spotlight, the four elements of the parallel light source are comprised in the near limit value (near), far limit value (far), inside limit angle (iang), and outside limit angle (oang). With the addition of the

- Spot (SPOT)

element, the total number of elements is five.

Therefore the parallel light source uses `NJS_MDATA_1` through `NJS_MDATA_4`, and the spot light uses `NJS_MDATA_1` through `NJS_MDATA_5`.

```
#define NJD_MTYPE_POS_0(1<<0)    /* Uses NJS_MKEY_F*/  
#define NJD_MTYPE_VEC_3    (1<<3) /* Uses NJS_MKEY_F*/  
#define NJD_MTYPE_TARGET_3(1<<6) /* Uses NJS_MKEY_F*/  
#define NJD_MTYPE_RGB_8    (1<<9) /* Uses NJS_MKEY_UI32*/  
#define NJD_MTYPE_INTENSITY_9(1<<10)/* Uses NJS_MKEY_F1*/  
#define NJD_MTYPE_SPOT_10 (1<<11)/* Uses NJS_MKEY_SPOT*/  
#define NJD_MTYPE_POINT_10(1<<12)/* Uses NJS_MKEY_F2*/
```

"Vec" stands for vector, and "Target" for the target position.

The common "3" in `NJD_MTYPE_VEC_3` and `NJD_MTYPE_TARGET_3`, and the common "10" in `NJD_MTYPE_SPOT_10` and `NJD_MTYPE_POINT_10` means that these cannot be used simultaneously.

Type setting examples

- Point light source

```
type=NJD_MTYPE_POS_0 | NJD_MTYPE_RGB_8 | NJD_MTYPE_INTENSITY_9 | NJD_MTYPE_POINT_11 ;
```

- Parallel light source (with direction as vector)

```
type=NJD_MTYPE_POS_0 | NJD_MTYPE_VEC_3 | NJD_MTYPE_RGB_8 | NJD_MTYPE_INTENSITY_9;
```

- Spot light source (with direction as target)

```
type=NJD_MTYPE_POS_0 | NJD_MTYPE_TARGET_3 | NJD_MTYPE_RGB_8  
| NJD_MTYPE_INTENSITY_9 | NJD_MTYPE_SPOT_10 ;
```

Interpolation method is specified with the upper 2 bits of "inp_fn". This is the same as for object motion.

9 Other Information

The start of the light file contains the following data, which contain alignment settings intended for SH:

```
?if USE_LIGHT_ALIGN  
?pragma USE_ALIGNDATA(LightName)  
?endif
```




6. NINJA LIGHT

1 How to set LIGHT

1.1 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

1.2 void njDeleteLight(NJS_LIGHT*)

Parameter:	NJS_LIGHT*ptr
Description:	This function deletes created Light ptr.
Return Value:	None
Remarks:	None

1.3 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.

1.4 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.

```
1.5 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.

1.6 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

1.7 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

```
1.8 void njSetLightAngle(NJS_LIGHT*, NJS_Angle, NJS_Angle)
```

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

1.9 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

1.10 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)

1.11 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

1.12 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)

1.13 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)

1.14 void njSetUserLight(NJS_LIGHT*, NJS_LIGHT_FUNC*)

Parameter:	NJS_LIGHT*ptr NJS_LIGHT_FUNC func
Description:	This function sets user setting light function func for Light ptr.
Return Value:	None
Remarks:	None

1.15 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

1.16 void njTranslateLightV(NJS_LIGHT*, NJS_VECTOR*)

Parameter:	NJS_LIGHT*ptr NJS_VECTOR *vctr
Description:	This function translates light matrix registered by njCreateLight in the direction of vector vctr.
Return Value:	None
Remarks:	None

1.17 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				

1.18 void njRotateLightX(NJS_LIGHT*, NJS_Angle)

Parameter:	NJS_LIGHT*ptr	NJS_Angleang
Description:	This function rotates light matrix registered by njCreateLight around X axis at ang angle.	
Return Value:	None	
Remarks:	None	

1.19 void njRotateLightXYZ(NJS_LIGHT*, NJS_Angle, NJS_Angle, NJS_Angle)

Parameter: NJS_LIGHT*ptr
NJS_Anglexang
NJS_Angleyang
NJS_Anglezang

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

Return Value: None

Remarks: None

1.20 void njRotateLightY(NJS_LIGHT*, NJS_Angle)

Parameter: NJS_LIGHT*ptr
NJS_Angleang

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

Return Value: None

Remarks: None

1.21 void njRotateLightZ(NJS_LIGHT*, NJS_Angle)

Parameter: NJS_LIGHT*ptr
NJS_Angleang

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

Return Value: None

Remarks: None

1.22 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)
```

1.23 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 function.

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

The other way for setting light structure is the way to use `njSetLight`. This way is used for the structure for which 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. Then if you want to stop calculation of light source for a certain model, you must set `njLightOff` before drawing model.

Please note that `njLightOff` and `njLightOn` keep current status.

Please refer to Reference for more detail information about functions, arguments, structures etc.

Example1) Sets Light `lt1` as spotlight and `Lightlt2` 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(...);

.....
}

```

Example 3) 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(...);

    .....
}
```


1.24 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]
[<i>specular</i>]	argb or rgb	intensity_spec
[<i>diffuse</i>]	argb or rgb	intensity_diff
[<i>ambient</i>]	(argb or rgb :pending)	intensity_amb
[<i>exponent</i>]	exp	None

<i>specular:</i>	highlight	Softimage sets 0 to 1 for nomal RGB value.
<i>diffuse:</i>	Normal light	Softimage sets 0 to 1 for nomal RGB value.
<i>ambient:</i>	Ambient light	Softimage sets 0 to 1 for nomal RGB value.
<i>exponent:</i>	Exponent for highlight	Softimage set 0 to 300 for nomal RGB value.

(We will not support HSV at Ninja Library.)

1.25 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_ATTRattr;(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
```

1.26 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
```

<lsrc>

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

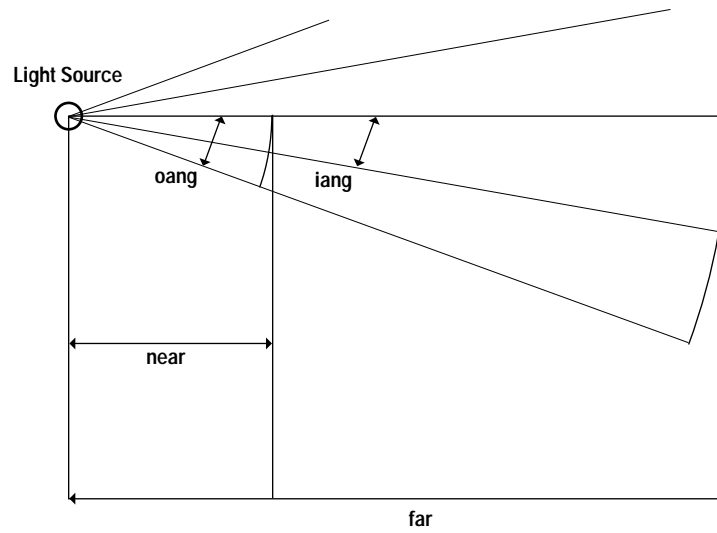


Figure 1.1 Spot light source.

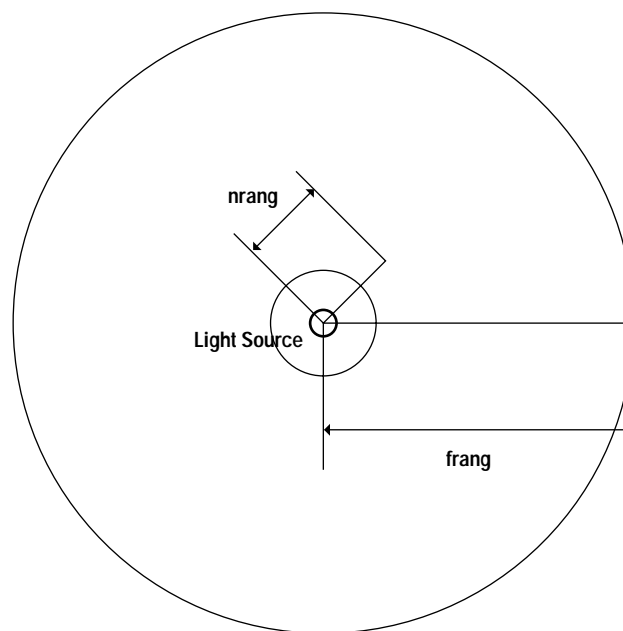


Figure 1.2 Point light source.

1.27 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 * fag)
    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      idev;      (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.)



7. Scroll Guide

1 Revision Information

1.1 Ver.0.04

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

1.2 Ver.0.05

* “3.2.5” and the description of the member clip in “5.3 Scroll-related Structure” were changed.

* “5.4 Color Definition” was modified.

2 Image Units as Related to Scrolling

2.1 Overview

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

2.2 Image Units

Pixel

The smallest component unit of an image

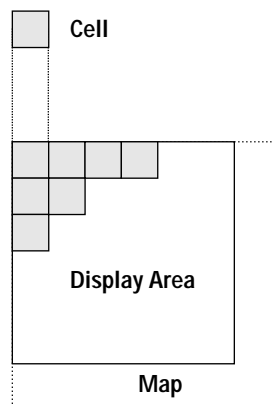
Cell

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`.



3 Scroll Rotation, Resizing, and Movement

3.1 Overview

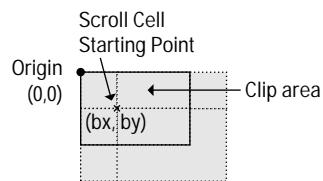
This chapter shows the meanings of the various values used in setting scroll displays and scroll structures, and how those values are calculated.

3.2 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).



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

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

The diagram shows the origin (0,0) at the top-left corner of the clip area. A point (-bx, -by) is marked within the clip area.

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

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

The diagram shows the origin (0,0) at the top-left corner of the clip area. A point (cx, cy) is marked within the clip area, labeled 'Center of rotation'.

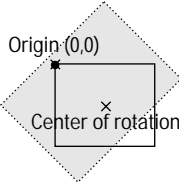
(4) Make the center of rotation into the origin, and rotate via the matrix m (see Figure)

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

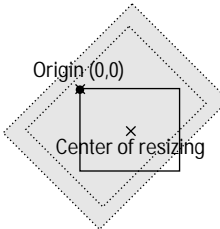
The diagram shows the origin (0,0) at the top-left corner of the clip area. A point (cx, cy) is marked within the clip area, labeled 'Center of rotation'. The clip area is rotated.

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

(6) Resize by s_x, s_y , centering on the center of resizing (spx, spy) (see Figure).

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


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

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


4 Scroll Programming

4.1 Overview

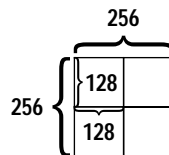
In this chapter, we cover everything from drawing of Cells to depicting Scroll area.

4.2 Example of Programming a Scroll

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.



Convert Cell Image to pvr format

Use tools to convert textures to PVR format.

Create Texture List

Create the texture name structure and texture list structure. Refer to “Texture Guide” for the detail on the way to create them.

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)}
};

```

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)
scl	applies color to entire scroll. Varies according to color mode

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 section 5)

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

Using the scroll functions, you can draw the scrolls

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

5 Color

5.1 Overview

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

5.2 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).

6 Scroll function, Structures, and Definitions

6.1 Overview

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

6.2 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.

6.3 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                /*Coordinatesfordrawingscrollorigin */
    */
    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;

```

6.4 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 muximum 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_TRANS41

```

6.5 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;
```



8. *Texture Guide*

1 Terminology

1.1 Overview

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

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 chapter2 for the detail.

Stride Value

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

2 Creating Textures

2.1 Overview

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

2.2 PVR Format

Global Index Tag	ID Area "GBIX"	4byte
	Byte Number to the Next Tag	4byte
	Global Index	4byte

PVR Format Tag	ID Area "PVRT"	4byte
	Byte Number to the Next Tag	4byte
	Texture Attribute	4byte
	Width	2byte
	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.

2.3 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				32					
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.

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

2.4 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.

	ARGB1555	RGB565	ARGB444	YUV422	Bump	ARGB8888
Twiddled	A	A	A	F	F	X
Twiddled MM	A	A	A	F	F	X
VQ	A	A	A	F	F	X
VQ MM	A	A	A	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

Table 1.1 Texture formats supported by NINJA

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

3 Memory

3.1 Overview

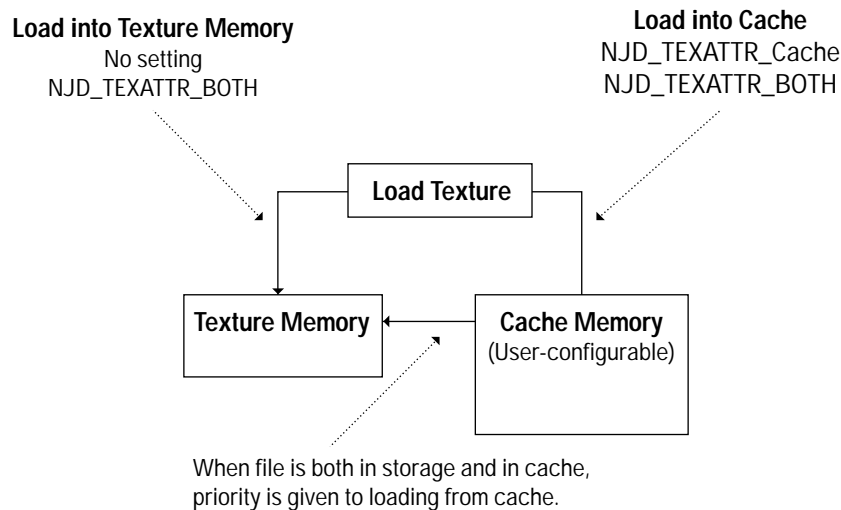
Ninja uses both texture memory and cache memory for loading textures. This chapter explains the two types of memory.

3.2 Texture Memory

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

3.3 Cache

In order to make effective use of the texture memory area, users can set the area where textures can be loaded on the main memory. This area is called “cache area”. 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.



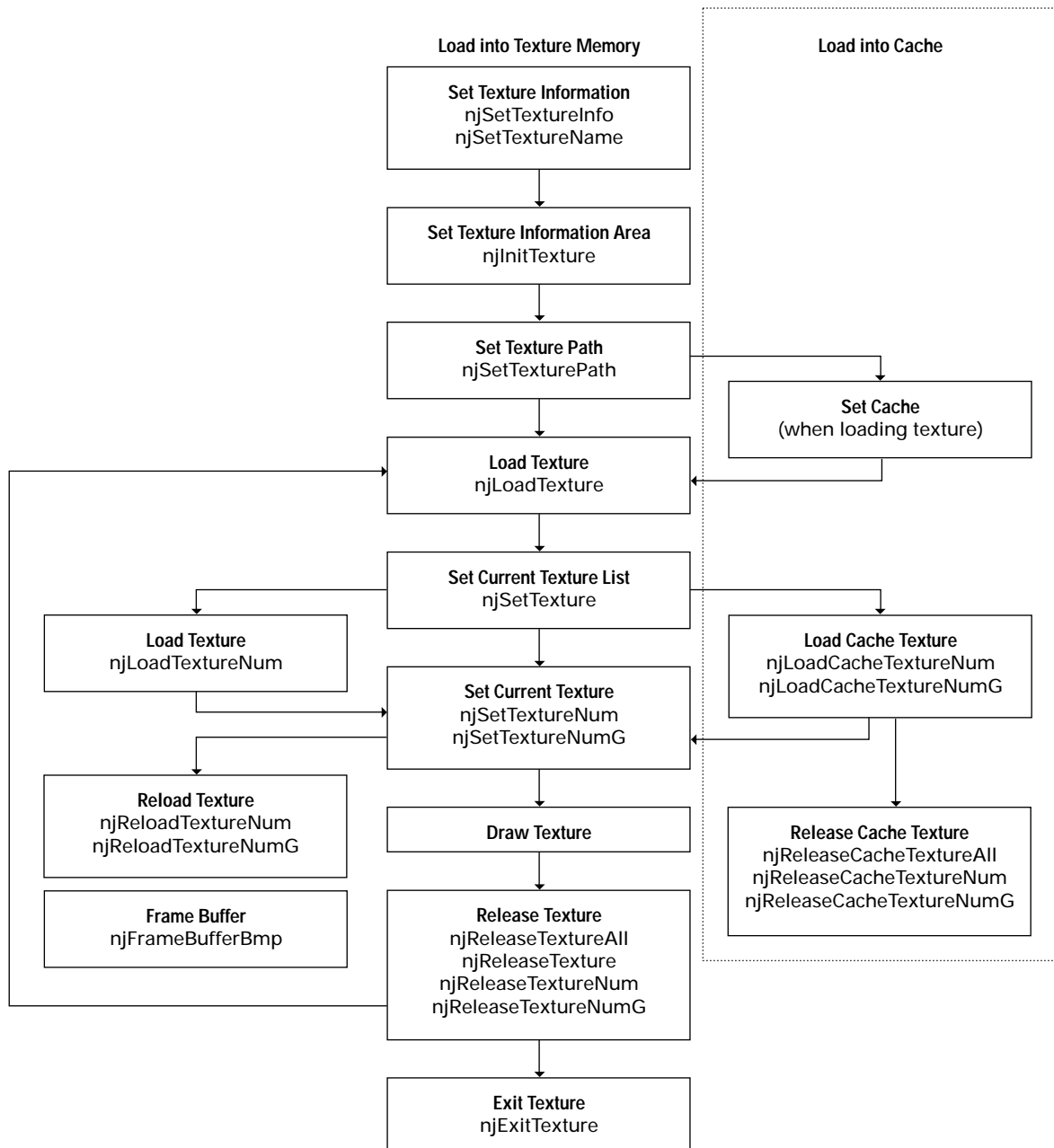
* Cache textures can not be used in SET4.

4 Loading Textures

4.1 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.

4.2 Flowchart of Texture Loading



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

Note: `njSetTexturePath` and `njFramBufferBmp` can not be used by target.

4.3 Setting a Texture Buffer

In Ninja for Set2, work buffer which is required when loading a texture can be obtained inside texture functions. After this, this work area is set by the following function.

`void njInitTextureBuffer (Sint8 *addr, Uint32 size)`

“addr” is the head pointer of texture work buffer and “size” is the size of work buffer.

About “size”, it becomes the biggest one in PVR files when loading from a file. When loading from memory, work buffer is not required for files which are conformed to PVR files. About targets of SET4 and over, in order to load files from CD, a unit of loading becomes 1 sector (2048Byte) and also about the buffer size, the numbers below 1 sector is raised to the next sector.

Work buffer is used only during executing `njLoadTexture` and `njLoadTextureNum` and is not used except this time. So it is OK to open it as soon as finished loading of textures.

Texture size of PVR files are as follows.

Table 1.2 TWIDDLED(GLOBALINDEX 12Byte, including the header 16Byte)

	SET2		SET4	
Size	MIPMAP	NO MIPMAP	MIPMAP	NO MIPMAP
8x8	0xC8	0x9C	0x800	0x800
16x16	0x2C8	0x21C	0x800	0x800
32x32	0xAC8	0x81C	0x1000	0x1000
64x64	0x2AC8	0x201C	0x3000	0x2800
128x128	0xAAC8	0x801C	0xB000	0x8800
256x256	0x2AAC8	0x2001C	0x2B000	0x20800
512x512	0xAAAC8	0x8001C	0xAB000	0x80800
1024x1024	0x2AAAC8	0x20001C	0x2AB000	0x200800

Table 1.3 VQ(GLOBALINDEX 12Byte, including the header 16Byte)

Size	SET2		SET4	
	MIPMAP	NO MIPMAP	MIPMAP	NO MIPMAP
8x8	0x832	0x82C	0x1000	0x1000
16x16	0x872	0x86C	0x1000	0x1000
32x32	0x972	0x91C	0x1000	0x1000
64x64	0xD72	0xC1C	0x1000	0x1000
128x128	0x1D72	0x181C	0x2000	0x2000
256x256	0x5D72	0x481C	0x6000	0x5000
512x512	0x15D72	0x1081C	0x16000	0x11000
1024x1024	0x55D72	0x4081C	0x56000	0x41000

Table 1.4 RECTANGLE,STRIDE(GLOBALINDEX 12Byte, including the header 16Byte)

Size	SET2		SET4	
	MIPMAP	NO MIPMAP	MIPMAP	NO MIPMAP
8x8	X	0x5C	X	0x800
8x16,16x8	X	0x11C	X	0x800
8x32,32x8	X	0x21C	X	0x800
8x64,64x8	X	0x41C	X	0x800
8x128,128x8	X	0x81C	X	0x1000
8x256,256x8	X	0x101C	X	0x1800
8x512,512x8	X	0x201C	X	0x2800
8x1024,1024x8	X	0x401C	X	0x4800
16x16,	X	0x21C	X	0x800
16x32,32x16	X	0x41C	X	0x800
16x64,64x16	X	0x81C	X	0x1000
16x128,128x16	X	0x101C	X	0x1800
16x256,256x16	X	0x201C	X	0x2800
16x512,512x16	X	0x401C	X	0x4800
16x1024,1024x16	X	0x801C	X	0x8800

	SET2		SET4	
32x32	X	0x81C	X	0x1000
32x64,64x32	X	0x101C	X	0x1800
32x128,128x32	X	0x201C	X	0x2800
32x256,256x32	X	0x401C	X	0x4800
32x512,512x32	X	0x801C	X	0x8800
32x1024,1024x32	X	0x1001C	X	0x10800
64x64	X	0x201C	X	0x2800
64x128,128x64	X	0x401C	X	0x4800
64x256,256x64	X	0x801C	X	0x8800
64x512,512x64	X	0x1001C	X	0x10800
64x1024,1024x64	X	0x2001C	X	0x20800
128x128	X	0x801C	X	0x8800
128x256,256x128	X	0x1001C	X	0x10800
128x512,512x128	X	0x2001C	X	0x20800
128x1024,1024x128	X	0x4001C	X	0x40800
256x256	X	0x2001C	X	0x20800
256x512,512x256	X	0x4001C	X	0x40800
256x1024,1024x256	X	0x8001C	X	0x80800
512x512	X	0x8001C	X	0x80800
512x1024,1024x512	X	0x10001C	X	0x100800
1024x1024	X	0x20001C	X	0x200800

4.4 Setting Cache Buffer

The cache area which has been obtained inside cache functions so far is now set by users as same as texture buffer. Being different from texture buffer, cache buffer requires total size all through the time for save. The cache size required for cache buffer equals to the total size after subtracting header size from each texture size.

```
void njInitCacheTextureBuffer(Sint8 *addr,Uint32 size)
```


4.5 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 (Modified)

Can not be used in SET4 and over.

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

NJD_TEXATTR_CACHE

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_TEXSURFACEtexsurface;

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;
UInt32	*pVirtual; <- New
UInt32	*pPhysical; <- New

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 textureImage
Category codeTWIDDLED
Color formatARGB1555
Size 256x256
*/
njSetTextureInfo(&Info,Image,NJD_TEXFMT_TWIDDLED|NJD_TEXFMT_ARGB_1555,256,256);
/* Set the file "file0.pvr" for texname[0] by GlobalIndex "0"*/
njSetTextureName(&texname[0],"file0.pvr",0,NJD_TEXATTR_TYPE_FILE);
/* Set the memory texture image for texname[1] by GlobalIndex "1"*/
njSetTextureName(&texname[1],&Info,1,NJD_TEXATTR_TYPE_MEMORY);
/* Setting texture buffer
(It is enough with 0x2001C but better to set a rather large number)*/
njInitTextureBuffer(buffer, 0x30000);
/* The initial of the texture */
njInitTexture(texmemlist,2);
/* Load texture*/
njLoadTexture(&texlist);
```

4.6 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:

```
NJS_TEXNAME texname[] = {    {"file0.pvr",...},    /* texture number 0 */
                           {"file1.pvr",...},    /* texture number 1 */
                           {"file2.pvr",...},    /* texture number 2 */
                           {"file3.pvr",...},    /* texture number 3 */
                           :
                           {"filen.pvr",...}};    /* *texture number n */
```

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

4.7 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.

4.8 Automatic allocation of Global Index Number

In NINJA Ver00040032 or over, textures without global index can be loaded. In this case, Global index allocates global index numbers in descending order (from 0xFFFFFFFFEF to 0xFFFFFFFFEE, 0xFFFFFFFFED...).

The initial value of global index used for automatic allocation can be set by the following function.

```
void njInitTextureGlobalIndex(UINT32 globalIndex);
```

Starting from the global index set by this function, global index numbers are allocated in the order of globalIndex, globalIndex-1, globalIndex-2...

As automatic allocation of Global Index heads for only in the descending order, even if a texture is deleted, the next number is allocated by global index. To set back the global index, please reset by njInitTextureGlobalIndex function.

Also, please note that if global index collides with the one set by the normal way, the one loaded earlier has priority over the other one.

4.9 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	tspparambuffer;	Data set by H/W<- New
UINT32	texparambuffer;	Data set by H/W <- New
UINT32	texaddr;	BIT_0: Load into texture memoryBIT_1: Load into cache
NJS_TEXINFO	texinfo;	Texture info structure
UINT16	count;	Use number of times
UINT16	dummy;	Error code (New addition)

In case that 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) //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) //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 NJD_TEXERR_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.

4.10 Memory Texture

Texture data expanded in main memory or texture data created in main memory can be loaded and used as texture data. The format which can be used as texture data is header information (global index tags, header tags) + data. Only the part of data can also be used as texture data. The following are setting method for each case.

In case that memory texture includes header information

Table 1.5 *Uint16 T009[]* = {

0x4247,	0x5849,	0x0004,	0x0000,	0x1d4c,	0x0000,	0x5650,	0x5452,
0x8008,	0x0000,	0x0101,	0x0000,	0x0080,	0x0080,	0xad20,	0xee1,
0xac40,	0xe5c0,	0xff41,	0xff21,	0xf5e0,	0xe580,	0xd580,	0xf700,
0xde00,	0xff80,	0xff40,	0fee0,	0ffe0,	0ffe0,	0xff21,	0xf720,

TWIDDLED Texture

RGB565

includes data of 128x128

?

};

Global index header part

PVR header part

Texture data part

```
NJS_TEXINFOinfo;                                Don't have to save info after loading textures.
NJS_TEXNAME texname[1];
NJS_TEXLIST texlist = {texname,1};

njSetTextureInfo(&info[0],T009,0,0,0);
njSetTextureName(&texname[0],&info[0],0,NJD_TEXATTR_TYPE_MEMORY);
```

In case that header information includes global index information, don't have to specify global index information as the 3rd and 4th argument of njSetTextureName function. Also, as there is PVR header information, don't have to set information to the 3rd, 4th and 5th argument of njSetTextreInfo function.

In case that memory texture does not include header information

Table 1.6 *Uint16 T009[]* = {

0xac40,	0xe5c0,	0xff41,	0xff21,	0xf5e0,	0xe580,	0xd580,	0xf700,
0xde00,	0xff80,	0xff40,	0fee0,	0ffe0,	0ffe0,	0xff21,	0xf720,

TWIDDLED Texture

RGB565

Includes data of 128x128

?

};

Texture data part

```
NJS_TEXINFOinfo;      Don't have to save info after loading textures
NJS_TEXNAME texname[1];
NJS_TEXLIST texlist = {texname,1};

njSetTextureInfo(&info,T009,NJD_TEXFMT_RGB_565|NJD_TEXFMT_TWIDDLED,128,128);
njSetTextureName(&texname[0],&info,0,NJD_TEXATTR_TYPE_MEMORY|NJD_TEXATTR_GLOBALINDEX);
```

In case that there is only data part of PVR format without both global index information and PVR header information, set global index information to the 3rd or 4th argument of njSetTextureName function, and set texture type and color format to the 3rd argument and set texture length and width to the 4th and 5th argument of njSetTextreInfo function.

4.11 Render Texture

When the category of the texture is STRIDE or RECTANGLE, instead of rendering to the usual frame buffer, rendering to a texture can be done. By using this texture, a user can do suspected environment mapping. As render texture is done by doing ren-dering to a texture and drawing again using it, if doing multiple render textures in one frame, the number of doing rendering increases as much as the number of doing render texture, and then the performance falls. If the texture which are specified by the render texture is smaller than the frame buffer, rendering is done as much as the number of the size of the texture started from the upper-left in the display. Also, the color mode of the texture which is used when doing render texture must be same as the color mode of the frame buffer. Render texture reserves just specified size in the texture area This is a difference of a render texture and a frame buffer texture.

Example

```
void njUserInit(void)
{
    /*In case of using render texture, let color modes of njInitSystem and a frame buffer
    same. */
    njInitSystem( NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB565, 1 );

    /* Obtain the dummy memory area for textures. */
    buff = njMalloc(0x8001C);
    /* Let the color same as a frame buffer. Set the size 512x512 */
    njSetTextureInfo(&info,buff,NJD_TEXFMT_STRIDE|NJD_TEXFMT_RGB_565,512,512);
    njSetTextureName(&texname[0],&info,0,NJD_TEXATTR_TYPE_MEMORY|
    NJD_TEXATTR_GLOBALINDEX);
    /* buff is required only during njLoadTexture */
    njInitTextureBuffer(buff,0x8001C);

    njInitTexture( tex, 100 );
    njLoadTexture(&texlist);
    /* After njLoadTexture, OK to open the area obtained as a dummy. */
    njFree(buff);
    /* Set the stride value to 512*/
    njSetRenderWidth(512);
}

Sint32 njUserMain(void)
{
    /* Draw models etc. */
    njDrawObject( OBJECT );

    njSetTexture(&texlist);
    /* Rendering to the texture number 0.
    As the size of this texture is 512x512, rendering is started from the upper-left for
    512x512. */
    njRenderTextureNum(0);

    /* Draw using texture to which rendering is done*/
    njDrawTexture( poly, 4, 0,TRUE);
}
```


5 Texture functions, Structures, and Definitions

5.1 Overview

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

5.2 Texture Functions

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.

njInitTextureBuffer (New function)

Set work buffer of a texture

Format

```
#include <Ninja.h>
void njInitTextureBuffer(addr,size);
Sint8* addr
Uint32 size
```

Parameter

*addr Head pointer of a workbuffer size
Work buffer size

Return Value

none

Function

Set required memory for work buffer of a texture. It is OK to open the memory set here after executing njLoadTexture or njLoadTextureNum.

Note

This function must be called before executing njLoadTexture or njLoadTextureNum,

`njInitCacheTextureBuffer` **(New Function)**

Set cache texture buffer

Format

```
#include <Ninja.h>
void njInitCacheTextureBuffer(addr,size);
Sint8* addr
Uint32 size
```

Parameter

*addr	The head pointer of cache texture buffer
size	Cache texture buffer size

Return Value

None

Function

Set required memory for the cache texture. Memory set here is required all through the time using the cache texture.

Note**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.

In SET5 or over, DMA is used when the address of texture buffer for transfer to texture memory becomes 32 byte align. Therefore, in case of using njLoadTexture with forbidding interrupt, be careful not to let the address be 32 byte align.

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, func-tion returns an error

Note

Before running this function, it is necessary to run njInitTexture and njSetTexture. In SET5 or over, DMA is used when the address of texture buffer for transfer to texture memory becomes 32 byte align.

Therefore, in case of using njLoadTextureNum with forbidding interrupt, be careful not to let the address be 32 byte align.

njSetTexture

Set current texture list

Format

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

Patameter

*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

njLoadCacheTexture

Load texture from cache memory to texture memory

Format

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

Parameter

*texlist	The pointer of NJS_TEXLIST structure
----------	--------------------------------------

Return Value

Success	1
Failure	-1

Function

Load texture in the texture list from cache memory into texture memory

Notes

Selected texture must be loaded into cache 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 loaded 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

Functions

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

Functions

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

Functions

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

Functions

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 re-leased, 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

Functions

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

Returning all free area in texture memory or maximum free block

Functions

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

Functions

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

Notes

Be sure to call this function when finished using textures.

njSetTexturePath(Can not be used in the target)

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

Functions

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	*infoTexture Information (output)
UInt16	*texPointer of memory texture
Sint32	TypeTexture type
Sint32	nWidthTexture width
Sint32	nHeightTexture 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	
NJD_TEXFMT_VQ_MM	
NJD_TEXFMT_PALETTE4	Not available
NJD_TEXFMT_PALETTE4_MM	Not available
NJD_TEXFMT_PALETTE8	Not available
NJD_TEXFMT_PALETTE8_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.

Set NJD_TEXATTR_TYPE_FILE to attr. For textures in PVR format, in case that global index is not used in files or there is no chunk of Global index in PVR format texture, set NJD_TEXATTR_TYPE_MEMORY to attr and set globalIndex to global index.

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 way to use.

njReloadTextureNum

Reload texture by texture number

Format

```
#include <Ninja.h>
```

```
Shint32 njReloadTextureNum(n,texaddr,attr,lod);
```

Parameters

Unit32	n	Current texture list texture number
Void	*texaddr	Filename or texture memory address
Unit32	attr	Texture attribute
Unit32	lod	Mip-map level

Return Value

Success	1
Failure	-1

Functions

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 x 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.

Notes

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

njReloadTextureNumG

Reload the global index number texture.

Format

```
#include <Ninja.h>
```

```
Shint32 njReloadTextureNumG(globalIndex, texaddr, attr, lod);
```

Parameters

Unit32	n	Global index texture number
Void	*texaddrFilename or texture memory address	
Unit32	attrTexture attribute	
Unit32	lodMip map level	

Return Value

Success	1
Failure	-1

Functions

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 x 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.

Notes

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

njRenderTextureNum

Do rendering to the texture area of the texture number.

Format

```
#include <Ninja.h>
```

```
void njSetRenderTextureNum(n);
```

Parameter

Uint32 n Texture number of the current texture list

Return Value

None

Functions

Do rendering to the texture of the number n in the current texture list.

The texture to which rendering can be done is NJD_TEXFMT_RECTANGLE or

NJD_TEXFMT_STRIDE. When the texture area is smaller than the display, do rendering to the area where is from the upper-left of the display to its coordinate.

In case of NJD_TEXFMT_STRIDE, the stride value must be set by njSetRenderWidth.

The stride value equals to the width of the rendering area usually.

Notes

Doing rendering to the texture area means that rendering is done twice adding up the usual rendering to the frame buffer.

njRenderTextureNumG

Do rendering to the texture area of the Global Index number.

Format

```
#include <Ninja.h>
```

```
void njRenderTextureNumG(globalIndex);
```

Parameter

Uint32 globalIndex GlobalIndexNumber

Return Value

None

Functions

Do rendering to the texture of which Global Index number is globalIndex

The texture to which rendering can be done is NJD_TEXFMT_RECTANGLE or NJD_TEXFMT_STRIDE.

When the texture area is smaller than the display, do rendering to the area where is from the upper-left of the display to its coordinate.

In case of NJD_TEXFMT_STRIDE, the stride value must be set by njSetRenderWidth.

The stride value equals to the width of the rendering area usually.

Notes

By doing rendering to the texture area, it becomes that rendering is done twice adding up the usual rendering to the frame buffer.

`njSetRenderWidth`

Set Stride value.

Format

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

Parameter

UInt32 nWidth	Stride value
---------------	--------------

Return Value

None

Functions

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.

Notes

`NjFrameBufferBmp` (Can not be used in the target)
Make a frame buffer into a bitmap.

Formats

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

Parameter

Unit8*filename	File name
----------------	-----------

Return Value

None

Functions

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)

Notes

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

NjFramebufferBmp2

Make a frame buffer into a bitmap. \

Formats

```
#include <Ninja.h>
void njFramebufferBmp2(buffer);
```

Parameter

char *buffer	Buffer for saving bitmap
--------------	--------------------------

Return Value

None

Functions

Makes a frame buffer into a 24-bit BMP. Buffer size can be calculated as shown below.

Buffer size (buffer) = Screen Length x Screen Width x 3 byte + 54 byte

Also, in case of using this function with the work area of njInitTextureBuffer, the following size is necessary.

Buffer size of njInitTextureBuffer =

Screen length x Screen width x Byte number of screen mode

* The byte number of screen mode becomes 2 byte when NJD_FRAMEBUFFER_MODE function is specified.

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)

If the buffer specified by this function is binary saved in Codescape, it becomes a bitmap.

Notes

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

Deleted functions

NjInitCache Texture

njLoadTextureNumG

5.3 Texture Structures

NJS_TEXSURFACE

```
typedef struct{
    Uint32      Type; /**/
    Uint32      BitDepth;/**/
    Uint32      PixelFormat;/**/
    Uint32      nWidth;/**/
    Uint32      nHeight;/**/
    Uint32      TextureSize;/**/
    Uint32      fSurfaceFlags;/**/
    Uint32      *pSurface;/**/
    Uint32      pVirtual;/**/ New member
    Uint32      pPhysical;/**/ New member
}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    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      tspparambuffer; /* TSPParambuffer */ New Member
    Uint32      tspparambuffer; /* TextureParambuffer */ New Member
    Uint32      texaddr;      /* texture Flag */
    NJS_TEXINFO texinfo;      /* texinfo */
    Uint16      count;        /* texture count */
    Uint16      dummy;        /* texture error */
} NJS_TEXMEMLIST;
```

5.4 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	Load from file
#define NJD_TEXATTR_TYPE_MEMORY	BIT_30	Load from memory
#define NJD_TEXATTR_TYPE_FRAMEBUFFER	BIT_28	Can not be used in SET4
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_ 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)
#define NJD_TEXFMT_VQ_MM             (0x0400)
#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)
```

Macro for getting texture data (New addition)

```
#define NJM_TEXTURE_WIDTH(texlist,n) \
(((NJS_TEXMEMLIST*)texlist->textures[(n)].texaddr)->texinfo.texsurface.nWidth)
#define NJM_TEXTURE_HEIGHT(texlist,n) \
(((NJS_TEXMEMLIST*)texlist->textures[(n)].texaddr)->texinfo.texsurface.nHeight)
#define NJM_TEXTURE_GLOBALINDEX(texlist,n) \
(((NJS_TEXMEMLIST*)texlist->textures[(n)].texaddr)->globalIndex)
```

6 Sample Program

6.1 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.

6.2 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:  Sint8 buffer[0x2B000];
9:
10: void njUserInit(void)
11:{
12:  njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1);
13:  /* Set two textures */
14:  njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE);
15:                                     NJD_TEXATTR_GLOBALINDEX);
16:  njSetTextureName(&texname[1], "file1.pvr", 1, NJD_TEXATTR_TYPE_FILE);
17:                                     NJD_TEXATTR_GLOBALINDEX);
18:  njInitTextureBuffer(buffer, 0x2B00);/* file0 and file1 are Twiddled Mipmap of 256x256*/
19:  njInitTexture(texmemlist, 2);
20:  njLoadTexture(&texlist); /* Load textures */
21:  njSetTexture(&texlist); /* Assignt texlist to Current texture list */
22:  /* Assign current texture to texture 0 of texlist*/
23:  njSetTextureNum(0);
24:
25:  /* Polygon data input */
26:  p[0].x = 100; p[0].y = 100;
27:  p[1].x = 200; p[1].y = 100;
28:  p[2].x = 200; p[2].y = 200;
29:  p[3].x = 100; p[3].y = 200;
30:  p[0].col.tex.u = 0; p[0].col.tex.v = 0;
31:  p[1].col.tex.u = 255; p[1].col.tex.v = 0;
```



```
32: p[2].col.tex.u = 255;p[2].col.tex.v = 255;
33: p[3].col.tex.u = 0;p[3].col.tex.v = 255;
34:}
35: Sint32 njUserMain(void)
36{
37: /* Draw polygon of texture */
38: njDrawPolygon2D(p,4,-100.f,NJD_FILL|NJD_USE_TEXTURE);
39: return NJD_USER_CONTINUE;
40:}
41: void njUserExit(void)
42:{
43: njExitTexture();
44: njExitSystem();
45:}
```

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: njSetTextureInfo(&Info,Image,NJD_TEXFMT_TWIDDLED|NJD_TEXFMT_ARGB_1555,256,256);
17: njSetTextureName(&texname[0], "file0.pvr",0,NJD_TEXATTR_TYPE_FILE);
18: njSetTextureName(&texname[1],&Info,1,NJD_TEXATTR_TYPE_MEMORY);
19: njInitTexture(texmemlist,2);
20: njLoadTexture(&texlist); /* Load texture */
21: njSetTexture(&texlist); /* Assign texlist to current texture list */
22: /* Assing texture 1 of texlis to current texture */
23: njSetTextureNum(1);
24:
25: /* Input plygon data */
26: p[0].x = 100; p[0].y = 100;
27: p[1].x = 200; p[1].y = 100;
```

```
28: p[2].x = 200; p[2].y = 200;
29: p[3].x = 100; p[3].y = 200;
30: p[0].col.tex.u = 0; p[0].col.tex.v = 0;
31: p[1].col.tex.u = 255; p[1].col.tex.v = 0;
32: p[2].col.tex.u = 255; p[2].col.tex.v = 255;
33: p[3].col.tex.u = 0; p[3].col.tex.v = 255;
34:}
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: Sint8 buffer[0x2B000];
9: Sint8 cbuffer[0x2AAAC*2];
10:
11: void njUserInit(void)
12:{
```

```
13: njInitSystem(NJD_RESOLUTION_VGA, NJD_FRAMEBUFFER_MODE_RGB555, 1)
14: njInitTexture(texmemlist, 2);
15: /* Set 2 textures */
16: njSetTextureName(&texname[0], "file0.pvr", 0, NJD_TEXATTR_TYPE_FILE |
17:                 NJD_TEXATTR_CACHE | NJD_TEXATTR_GLOBALINDEX);
18: njSetTextureName(&texname[1], "file1.pvr", 1, NJD_TEXATTR_TYPE_MEMORY |
19:                 NJD_TEXATTR_CACHE | NJD_TEXATTR_GLOBALINDEX);
20: njInitTextureBuffer(buffer, 0x2B000);
21: njInitCacheTextureBuffer(cbuffer, 0x2AAAC*2);
22: njLoadTexture(&texlist); /* Load textures */
23: njSetTexture(&texlist); /* Specify texlist to current texture */
24: njLoadCacheTextureNum(0); /* Load texture of number 0 from cache */
25: njLoadCacheTextureNum(1); /* Load texture of number 1 from cache */
26: /* Assign texture 0 of texlist to current texture */
27: njSetTextureNum(0);
28:
29: /* Polygon data input */
30: p[0].x = 100; p[0].y = 100;
31: p[1].x = 200; p[1].y = 100;
32: p[2].x = 200; p[2].y = 200;
33: p[3].x = 100; p[3].y = 200;
34: p[0].col.tex.u = 0; p[0].col.tex.v = 0;
35: p[1].col.tex.u = 255; p[1].col.tex.v = 0;
36: p[2].col.tex.u = 255; p[2].col.tex.v = 255;
37: p[3].col.tex.u = 0; p[3].col.tex.v = 255;
38;}
39:
40: Sint32 njUserMain(void)
41{
42: /* Draw texture polygon */
43: njDrawPolygon2D(p, 4, -100.f, NJD_FILL | NJD_USE_TEXTURE);
44: return NJD_USER_CONTINUE;
45;}
46:
47: void njUserExit(void)
48{
49: njExitTexture();
50: njExitSystem();
51;}
```

7 Notes for Texture functions

7.1 Overview

This chapter contains notes for using texture functions.

7.2 Notes for Switchover from SET2 to SET4/SET5

- 1) In SET2, the work area for texture functions were allocated inside. But in SET4 or over, please get texture buffer using `njInitTextureBuffer`. Refer to "4.3 Setting Texture Buffer" for the necessary size.
- 2) `njSetTexturePath` function can not be used in SET4 or over. Modify the part where `njSetTexturePath` function is used as follows.

SET2:

```
njSetTexturePath("\\image0");  
njLoadTexture(&texlist0);  
njSetTexturePath("\\image1");  
njLoadTexture(&texlist1);
```

SET4:

```
gdFsChangeDir("IMAGE0");  
njLoadTexture(&texlist0);  
gdFsChangeDir("../");  
gdFsChangeDir("IMAGE1");  
njLoadTexture(&texlist1);
```

- 3) As DMA transfer is unsupported in SET4, textures can not be loaded during rendering. Therefore, cache texture and reload texture can not be used during rendering.
- 4) `njFrameBufferBmp` function can not be used in SET4 or over. Please substitute `njFrameBufferBmp2` for `njFrameBufferBmp`.

7.3 Notes for using texture functions in SET5

In SET5, transfer from main memory to texture memory is DMA transfer when

the head address of the buffer is 32 byte alignment. When the head address is other than 32 byte alignment, transfer becomes CPU transfer.

In case of executing functions which are transferred to texture memory with the situation of forbidding interrupt, if the buffer is one of 32 byte alignment, DMA end interrupt is ignored. Be careful not to let the buffer 32 byte alignment.



9. Chunk Model Specifications

1 Overview

Ninja supports two format models, called the Basic Model and the Chunk Model. While a drawing function is executed in the Chunk Model, the data are placed in a continuous memory space so as to maintain integrity of the SH4 cache. Expandability, flexibility, and data expression efficiency are excellent. In future, further tuning will be carried out, centering on the Chunk Model. The Basic Model is supported, but does not include the new features.

In the Chunk Model, the model structure contents have been significantly changed. The object structure is not changed, except for the fact that the model structure pointers have been altered to the Chunk Model.

Motions and textures besides the model use the same format as before. However, for compatibility with camera and light, the format of structure members has been changed.

For information on the Basic Model and the texture structure, refer to the Basic Model Specifications.

The features of the Chunk Model are listed below.

1.1 Chunk Model Features

Based on triangular strip drawing. Currently, triangular, quadrilateral, and N-sided polygon drawing is not supported. Performance has topmost design priority. The data consist of the vertex list "vlist" and polygon list "plist". Data are arranged on "vlist" and "plist" in IFF chunk format for keeping the memory area uniform and for protecting the cache during drawing execution. Both the polygon side and vertex side can contain vertex color information. On the polygon side, the vertex can be assigned an individual color for each polygon.

The polygon side can have a vertex normal line. Because the vertex normal line is in polygon units, the "softimage" vertex normal line can be output as is. Discontinuity data can also be output. The polygon side and the vertex side can have a user flag area (max. 16 bit x 3 for the polygon side and 32 bit x 1 for the vertex side). Currently, this area is used when outputting vertex color data to the user flag area. In future, tools for writing user data to this area are planned. The material is stored in "plist", and only the difference to previous material (differential settings) are updated. This reduces the number of material settings compared to the Basic Model. Material can be deleted at the time of converter output. When drawing an identical model (for example a tree), data can be optimized by deleting all material and making external user settings.

10-bit normal lines are supported, to allow a reduction in data volume. The XYZ normal lines are stored in Uint32 with 10 bits each. 2 bits are filled with 0 as reserved area.

Collision data Chunk Volume output is supported. Triangular, quadrilateral, and triangular strip output is possible, without material information. A user flag area is possible. Currently, the material color is output in this area. Separate triangular Chunk Volume (volume3) can also be used as modifier volume. When "volume34" is specified for the converter, a quadrilateral shape is created from triangles adjoining at 0.1 degree angles. 3D Studio MAX can only output triangular data, but in this case, quadrilateral collision data can be generated.

To use the SH4 hardware efficiently and allow high-speed processing, the vertex format is supported. (NJD_CV_SH, NJD_CV_VN_SH). High speed is achieved by using the SH4 matrix processing commands efficiently. This is used when performance has priority over data volume.

The following two types of UV value expressions are available: 0-255 UVN, and high-resolution 0-1023 UVH. UVN is a conventional expression which has been used in Basic a Model. However, resolution suffers at sizes exceeding 256. With UVH and high-resolution mode, 1024 x 1024 texture can be specified in 1-pixel units. But compared to UVN, the texture repeat count of UVH decreases proportionally to the increase in resolution (32 times for UVH vs. 128 times for UVN). UVN and UVH can be switched by convert option for the whole of model tree and also material names can be used to switch at each model unit. In case that the UV value is specified by material name, UV value expressions are changed by setting to only one material (among some materials used for single model). The default is UVN.

2 Model Structures

2.1 Structure Diagram

Chunk Object Tree

Structure Description

```
Float, Angle
typedef float Float          /* Floating-point operation type      */
typedef Sint32 Angle         /* Angle of rotation          */
For angles, 0x0000 - 0xFFFF correspond to 0 to 360 degrees.
```

Point structure

```
typedef struct {
Float   x;          /* X value          */
Float   y;          /* Y value          */
Float   z;          /* z value          */
} NJS_POINT3, NJS_VECTOR;
Gives the vertex XYZ values.
```

Chunk Model structure

```
typedef struct {
Sint32   *vlist;          /* Vertex chunk list      */
Sint16   *plist;          /* Polygon chunk list     */
NJS_POINT3 center;        /* Model center           */
Float    r;              /* Model diameter         */
} NJS_CNK_MODEL;
```

"vlist" contains the vertex list data as a Sint32 array in "iff" chunk format.

"plist" contains polygon index list as a Sint16 array in "iff" chunk format.

"center" specifies the exterior circumference center of the model, with the radius "r".

Chunk object structure

```
typedef struct cnkobj {
Uint32   evalflags; /* Matrix processing evaluation flag*/
NJS_CNK_MODEL *model; /* Chunk model pointer*/
Float    pos[3]; /* Motion amount */
Angle    ang[3]; /* Rotation amount*/
Float    scl[3]; /* Scale          */
struct cnkobj *child; /* Child pointer*/
struct cnkobj *sibling; /* Sibling pointer*/
} NJS_CNK_OBJECT;
```

Gives the child/parent structure of the model. The evalflags contain flags for matrix processing optimization, and a chunk model structure pointer is hooked to the model. For nodes without polygons, this pointer is set to NULL. "pos" specifies the amount of position motion, and "rot" specifies the rotation amount. "scl" specifies the scale and "child" and "sibling" supply the child and sibling pointers.

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 expected by LightWave3D*/
#define NJD_EVAL_SKIP        BIT_6 /* Skip motion              */
#define NJD_EVAL_SHAPE_SKIP BIT_7
                                /* Skip shape motion        */
#define NJD_EVAL_MASK        0xff
                                /* Mask for extracting above bits */
```

These flags are set by the converter.

NJD_EVAL_UNIT_POS is set when the parallel motion amount is "0". Parallel motion matrix processing is omitted when this flag is set.

NJD_EVAL_UNIT_ANG is set when the rotation angle is "0". Rotation matrix processing is omitted when this flag is set.

NJD_EVAL_UNIT_SCL is set when the scale is "1" for x, y, and z. Scale matrix processing 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 processing steps are omitted, and the matrix "push pop" operation is 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 "XYZ" 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 processing sequence is changed to "ZXY".

The NJD_EVAL_SKIP flag indicates that this node does not include motion data. During motion execution, matrix processing is carried out using the object structure value without incrementing the motion node, and then proceeds to the next node. This allows motion also with polygon models having a different configuration, provided that the bone structure is the same.

The NJD_EVAL_SHAPE_SKIP flag indicates that this node does not include shape motion data.

NJD_EVAL_SKIP and NJD_EVAL_SHAPE_SKIP can be specified by material names.

3 Chunk Specifications

3.1 Chunk Types

Chunk name	Symbol	Size	Description
Chunk NULL	CN	16bit	Long word alignment matching.
Chunk End	CE	16bit	Chunk data list end marker.
Chunk Bits	CB	16bit	Flag setting for Blend Alpha etc.
Chunk Tiny	CT	32bit	Flag and single value setting for TexId etc.
Chunk Material	CM	Variable	Diffuse, Specular, Exponent, Ambient setting.
Chunk Vertex	CV	Variable	Supplies vertex list.
Chunk Volume	CO	Variable	Supplies collision and modifier volume data.
Chunk Strip	CS	Variable	Supplies strip data.

Based on the basic structure, define a simplified chunk (Bits, Tiny, etc.) suitable for the respective purpose. The Chunk Vertex for the vertex is stored in the "vlist" for the Chunk Model structure. Other chunks are stored in "plist". These are defined by "NinjaCnk.h".

3.2 Chunk Structure

The basic chunk structure is as follows.

Chunk Vertex

[headbits(15-8) | ChunkHead(7-0)][longsize(15-0)][data]

Other than Chunk Vertex

[headbits(15-8) | ChunkHead(7-0)][shortsize(15-0)][data]

The ChunkHead supplies the function table entry number for that chunk. The library selects a function from this number for execution. By dividing processing functions into chunks, draw routines are simplified and can execute faster. Because the maximum table size is 256 entries, the upper 8 bits are available. These 8 bits (called headbits) are used for storing a part of the attribute flags according to the chunk type and purpose, for more efficient use of the data size. Note that the upper 8 bits must be masked when obtaining the function entry number of the chunk table.

The "shortsize" and "longsize" gives the offset until the start of the next chunk.

Usually, in iff format, the data offset until the next chunk is given by byte unit. But as plist which becomes object is the short arrangement and vlist is long arrangement, each offset is expressed at shortsize(2bytes) unit and longsize(4bytes) unit. It has the effect to enlarge the maximum expressible number of the offset until the next chunk. For example, when the user wants to rewrite only the material, it is possible to use "shortsize" to skip data until the material chunk is found, and then to overwrite the material value obtained in this way.

3.3 Chunk NULL

ChunkName : 'NJD_CN'

(Chunk NULL)

Outline:

This chunk consists only of the ChunkHead (16 bits). It is inserted between chunks for long word alignment matching in the "plist".

Format:

[ChunkHead(15-0)]

ChunkHead:

NJD_CN

Description:

```
#define NJD_CN (NJD_NULLOFF+0)
```

"plist" is a primary array based on Sint16. Therefore the strip end is not always a Sint32 boundary. Although the Chunk Material is a Sint16 array, reading it as Sint32 data will improve efficiency. For this purpose, the NJD_CN is added to the end of the Chunk Strip to create a Sint32 boundary. For performance reasons, boundary matching by library collating is still under evaluation. If boundary matching is not performed, NJD_CN is not used.

3.4 Chunk End

ChunkName : 'NJD_CE'

(Chunk End)

Outline:

This chunk consists only of the ChunkHead (16 bits). It specifies the end of the "plist" and "vlist" chunk list. For "vlist", it is treated as a ChunkHead (32 bits). The actual value is queried by checking whether the highest bit is raised to "1".

Format:

plist: [ChunkHead(15-0)]?16 bits chunk?

vlist: [ChunkHead(31-0)]?32 bits chunk?

ChunkHead:

NJD_CE

Description:

```
#define NJD_CE (NJD_ENDOFF+0)
```

etects the end of the list.

3.5 Chunk Bits

Chunk bits are used for rewriting flags such as the attribute flag.

[headbits(15-8) | ChunkHead(7-0)](16 bits chunk)

The flags are stored in the upper 8 bits, and the lower 8 bits supply the chunk number. This chunk consists only of the ChunkHead (16 bits). The actual Chunk Bits are explained below.

ChunkName : 'NJD_CB_BA'

(Chunk Bits Blend Alpha)

Outline:

Sets "Blend Alpha" for the attribute flag in "plist".

Format:

[headbits(13-8) | ChunkHead(7-0)]

headbits:

13-11 = SRC Alpha Instruction(3bit)

10- 8 = DST Alpha Instruction(3bit)

ChunkHead:

NJD_CB_BA

Description:

#define NJD_CB_BA (NJD_BITSOFF+0)

"Blend Alpha" is set in two ways. As headbits for Chunk Material (see below), it can be set to "diffuse", "specular", and "ambient", and it can also be set to "Blend Alpha". To set only "Blend Alpha" without changing the material, use NJD_CB_BA.

The blending function combines the two RGBA values SRC and DST as shown below, and writes the result back to DST.

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

DST * BlendFunction(DST Alpha Instruction)

To the Blend function (instruction), a 3-bit instruction is input together with SRC/DST color. For each RGBA value, a coefficient weighted with four alpha values is returned.

Instruction	Field Value	Values Returned
Zero	0	(0, 0, 0, 0)
One	1	(1, 1, 1, 1)
'Other' Colour	2	(OR, OG, OB, OA)
Inverse 'Other' Colour	3	(1 - OR, 1 - OG, 1 - OB, 1 - OA)
SRC Alpha	4	(SA, SA, SA, SA)
Inverse SRC Alpha	5	(1 - SA, 1 - SA, 1 - SA, 1 - SA)
DST Alpha	6	(DA, DA, DA, DA)
Inverse DST Alpha	7	(1 - DA, 1 - DA, 1 - DA, 1 - DA)

"Other Color" and "Inverse Other Color" indicate that the DST color is used when specified for the SRC instruction, and the SRC color when specified for the DST instruction.

The abbreviations have the following meanings.

ZER:	Zero
ONE:	One
OC:	'Other' Color
IOC:	Inverse 'Other' Color
SA:	Src Alpha
ISA:	Inverse SRC Alpha
DA:	DST Alpha
IDA:	Inverse DST Alpha

Flag Blending Src?

```

#define NJD_FBS_SHIFT      11
#define NJD_FBS_ZER        0<<NJD_FBS_SHIFT)
#define NJD_FBS_ONE        (1<<NJD_FBS_SHIFT)
#define NJD_FBS_OC         (2<<NJD_FBS_SHIFT)
#define NJD_FBS_IOC        (3<<NJD_FBS_SHIFT)
#define NJD_FBS_SA         (4<<NJD_FBS_SHIFT)
#define NJD_FBS_ISA        (5<<NJD_FBS_SHIFT)
#define NJD_FBS_DA         (6<<NJD_FBS_SHIFT)
#define NJD_FBS_IDA        (7<<NJD_FBS_SHIFT)

#define NJD_FBS_MASK        (0x7<<NJD_FBS_SHIFT)

```

Flag Blending Dst?

```
#define NJD_FBD_SHIFT      8
#define NJD_FBD_ZER        (0<<NJD_FBD_SHIFT)
#define NJD_FBD_ONE        (1<<NJD_FBD_SHIFT)
#define NJD_FBD_OC         (2<<NJD_FBD_SHIFT)
#define NJD_FBD_IOC        (3<<NJD_FBD_SHIFT)
#define NJD_FBD_SA         (4<<NJD_FBD_SHIFT)
#define NJD_FBD_ISA        (5<<NJD_FBD_SHIFT)
#define NJD_FBD_DA         (6<<NJD_FBD_SHIFT)
#define NJD_FBD_IDA        (7<<NJD_FBD_SHIFT)

#define NJD_FBD_MASK        (0x7<<NJD_FBD_SHIFT)
```

ChunkName : 'NJD_CB_DA'

(Chunk Bits 'D' Adjust)

Outline:

Sets the mipmap 'D' adjust value for "plist".

Format:

[headbits(11-8) | ChunkHead(7-0)]

headbits:

11- 8 = Mipmap 'D' adjust(4)

ChunkHead:

NJD_CB_DA

Description:

```
#define NJD_CB_DA          (NJD_BITSOFF+1)
```

Adjusts the mipmap switching depth. The default is 1.00. This chunk should not be changed frequently. It is designed to suppress excessive mipmap switching.

Flag 'D' Adjust?

```
#define NJD_FDA_SHIFT      8
#define NJD_FDA_025        (1<<NJD_FDA_SHIFT)    /* 0.25 */
#define NJD_FDA_050        (2<<NJD_FDA_SHIFT)    /* 0.50 */
#define NJD_FDA_075        (3<<NJD_FDA_SHIFT)    /* 0.75 */
#define NJD_FDA_100        (4<<NJD_FDA_SHIFT)    /* 1.00 */
#define NJD_FDA_125        (5<<NJD_FDA_SHIFT)    /* 1.25 */
#define NJD_FDA_150        (6<<NJD_FDA_SHIFT)    /* 1.50 */
#define NJD_FDA_175        (7<<NJD_FDA_SHIFT)    /* 1.75 */
#define NJD_FDA_200        (8<<NJD_FDA_SHIFT)    /* 2.00 */
#define NJD_FDA_225        (9<<NJD_FDA_SHIFT)    /* 2.25 */
#define NJD_FDA_250        (10<<NJD_FDA_SHIFT)   /* 2.25 */
#define NJD_FDA_275        (11<<NJD_FDA_SHIFT)   /* 2.25 */
#define NJD_FDA_300        (12<<NJD_FDA_SHIFT)   /* 3.00 */
#define NJD_FDA_325        (13<<NJD_FDA_SHIFT)   /* 3.25 */
#define NJD_FDA_350        (14<<NJD_FDA_SHIFT)   /* 3.50 */
#define NJD_FDA_375        (15<<NJD_FDA_SHIFT)   /* 3.75 */

#define NJD_FDA_MASK        (0xf<<NJD_FDA_SHIFT)
```

ChunkName : 'NJD_CB_EXP'

(Chunk Bits Exponent)

Outline:

Sets the Exponent for the "plist" "specular". Values from 0 to 16 are valid.

Format:

[headbits(12-8) | ChunkHead(7-0)]

headbits:

12- 8 = Exponent(5) range:0-16

ChunkHead:

NJD_CB_EXP

Description:

```
#define NJD_CB_EXP (NJD_BITSOFF+2)
```

The exponent is set in two ways. To set "specular" in Chunk Material (see below), the upper 8 bits of the specular component in the Chunk Material are used to set the exponent (which becomes ERGB8888). To change only the exponent without changing the previously set exponent, use NJD_CB_EXP.

Flag EXPonent(range 0-16)?

```
#define NJD_FEXP_SHIFT      8
#define NJD_FEXP_00        (0<<NJD_FEXP_SHIFT)    /* 0.0 */
#define NJD_FEXP_01        (1<<NJD_FEXP_SHIFT)    /* 1.0 */
#define NJD_FEXP_02        (2<<NJD_FEXP_SHIFT)    /* 2.0 */
#define NJD_FEXP_03        (3<<NJD_FEXP_SHIFT)    /* 3.0 */
#define NJD_FEXP_04        (4<<NJD_FEXP_SHIFT)    /* 4.0 */
#define NJD_FEXP_05        (5<<NJD_FEXP_SHIFT)    /* 5.0 */
#define NJD_FEXP_06        (6<<NJD_FEXP_SHIFT)    /* 6.0 */
#define NJD_FEXP_07        (7<<NJD_FEXP_SHIFT)    /* 7.0 */
#define NJD_FEXP_08        (8<<NJD_FEXP_SHIFT)    /* 8.0 */
#define NJD_FEXP_09        (9<<NJD_FEXP_SHIFT)    /* 9.0 */
#define NJD_FEXP_10        (10<<NJD_FEXP_SHIFT)   /* 10.0 */
#define NJD_FEXP_11        (11<<NJD_FEXP_SHIFT)   /* 11.0 */
#define NJD_FEXP_12        (12<<NJD_FEXP_SHIFT)   /* 12.0 */
#define NJD_FEXP_13        (13<<NJD_FEXP_SHIFT)   /* 13.0 */
#define NJD_FEXP_14        (14<<NJD_FEXP_SHIFT)   /* 14.0 */
#define NJD_FEXP_15        (15<<NJD_FEXP_SHIFT)   /* 15.0 */
#define NJD_FEXP_16        (16<<NJD_FEXP_SHIFT)   /* 16.0 */

#define NJD_FEXP_MASK      (0x1f<<NJD_FEXP_SHIFT)
```

3.6 Chunk Tiny

Chunk Tiny is used to set the flag and one value. "TexId" corresponds to this. The size is fixed to 32 bits, consisting of the 16-bit Chunk Head and a 16-bit constant.

[headbits(15-8) | ChunkHead(7-0)][value(15-0)] (32 bits chunk)

Currently, only "TexId" and `NJD_CT_TID` which sets the texture function attribute flag are defined for Chunk Tiny.

ChunkName : 'NJD_CT_TID'

(Chunk Tiny TexId)

Outline:

Sets "TexId" (entry number in "TexList") in "plist".

Format:

[headbits(15-8) | ChunkHead(7-0)][texbits(15-13) | TexId(12-0)]

headbits:

15-14	= FlipUV(2)
13-12	= ClampUV(2)
11- 8	= Mipmap 'D' adjust(4)

ChunkHead:

`NJD_CT_TID`

texbits:

15-14	= Filter Mode(2)
13	= Super Sample(1)

TexId:

0?8191 (TexId Max = 8191)

Description:

```
#define NJD_CT_TID (NJD_TINYOFF+0)
```

Sets the texture function attribute flag and "TexId". By controlling the timing for texture switching, effective texture switching can be achieved. Because the number of bits allocated to "TexId" is 13, the maximum value is 8191.

Flag FLip (headbits)?

```
#define NJD_FFL_U (BIT_15)
#define NJD_FFL_V (BIT_14)
```

Controls the UV value flip.

Flag CLamp (headbits)?

```
#define NJD_FCL_U          (BIT_13)
#define NJD_FCL_V          (BIT_12)
```

Controls the UV value clamp.

Flag Filter Mode (texbits):

```
PS      :   Point Sampled
BF      :   Bilinear Filter?default?
TF      :   Tri-liner Filter
```

```
#define NJD_FFM_SHIFT      14
#define NJD_FFM_PS         ( 0<<NJD_FFM_SHIFT)
#define NJD_FFM_BF         1<<NJD_FFM_SHIFT)
#define NJD_FFM_TF         ( 2<<NJD_FFM_SHIFT)

#define NJD_FFM_MASK       ( 0x3<<NJD_FFM_SHIFT)
```

Controls the filter ring mode.

Flag Super Sample (texbits):

```
#define NJD_FSS            (BIT_13)
```

Controls super sampling.

```
#define NJD_TID_MASK       ( ~(NJD_FSS | NJD_FFM_MASK) )
```

3.7 Chunk Material

Chunk Material includes the "diffuse", "specular", and "ambient" setting. Only a difference to the previous value is set. If the headbits of the Chunk Strip (see below) contain the flag to disregard the "specular" value (NJD_FST_IL), the "specular" setting is omitted. If the flag to disregard the "ambient" value (NJD_FST_IA) is included, the "ambient" setting is omitted. "Blend Alpha" is set in the headbits (upper 8 bits). This part is equivalent to NJD_CB_BA. Details on "Blend Alpha" are given in the section on NJD_CB_BA. The upper 8 bits of the "specular" setting are used to specify the exponent. This part is functionally equivalent to NJD_CB_EXP. For NJD_CB_EXP, the flag bits 0 - 16 (NJD_FEXP_*) were used for the setting, whereas Chunk Material "specular" sets the 0 - 16 values directly.

[headbits(13-8) | ChunkHead(7-0)][shortsize(15-0)][Data]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

```
NJD_CM_D, NJD_CM_A, NJD_CM_DA,  NJD_CM_S,
NJD_CM_DS, NJD_CM_AS,  NJD_CM_DAS
```

Data:

Diffuse?ARGB8888

Specular?ERGB8888?E?exponent 0?16?

Ambient?NRGB8888?N?NOOP?

ChunkName : 'NJD_CM_D'

(Chunk Material Diffuse)

Outline:

Sets the "diffuse" value in "plist". The alpha value is stored in the upper 8 bits of the "diffuse" setting.

Format:

[headbits(13-8) | ChunkHead(7-0)][2(shortsize)][ARGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_D

ARGB:

ARGB8888

Description:?

```
#define NJD_CM_D (NJD_MATOFF+1)
```

Sets only "diffuse" for "Blend Alpha". The "specular" and "ambient" settings are kept at the current values.

ChunkName : 'NJD_CM_A'

(Chunk Material Ambient)

Outline:

Sets the "ambient" value in "plist". The upper 8 bits of the "ambient" setting contain 255 as a dummy value (NOOP).

Format:

[headbits(13-8) | ChunkHead(7-0)][2(shortsize)][NRGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_A

NRGB:

NRGB8888?N?NOOP 255?

Description:

```
#define NJD_CM_A (NJD_MATOFF+2)
```

Sets only "ambient" for "Blend Alpha". The "diffuse" and "specular" settings are kept at the current values.

ChunkName : 'NJD_CM_DA'

(Chunk Material Diffuse and Ambient)

Outline:

Sets the "diffuse" and "ambient" values in "plist". The upper 8 bits of the "diffuse" setting contain the alpha value. The upper 8 bits of the "ambient" setting contain 255 as a dummy value (NOOP).

Format:

[headbits(13-8) | ChunkHead(7-0)][4(shortsize)][ARGB][NRGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_DA

ARGB:

ARGB8888

NRGB:

NRGB8888?N?NOOP 255?

Description:

```
#define NJD_CM_DA (NJD_MATOFF+3)
```

Sets "diffuse" and "ambient" for "Blend Alpha". The "specular" setting is kept at the current value.

ChunkName : 'NJD_CM_S'

(Chunk Material Specular)

Outline:

Sets the "specular" value in "plist". The upper 8 bits of the "specular" setting contain the exponent (0 - 16).

Format:

[headbits(13-8) | ChunkHead(7-0)][2(shortsize)][ERGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_S

ERGB:

ERGB8888(E?Exponent 0?16)

Description:

```
#define NJD_CM_S (NJD_MATOFF+4)
```

Sets only "specular" for "Blend Alpha". The "diffuse" and "ambient" settings are kept at the current values.

ChunkName : 'NJD_CM_DS'

(Chunk Material Diffuse and Specular)

Outline:

Sets the "diffuse" and "specular" values in "plist". The upper 8 bits of the "diffuse" setting contain the alpha value. The upper 8 bits of the "specular" setting contain the exponent (0 - 16).

Format:

[headbits(13-8) | ChunkHead(7-0)][4(shortsize)][ARGB][ERGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_DS

ARGB:

ARGB8888

ERGB:

ERGB8888(E?Exponent 0?16)

Description:?

```
#define NJD_CM_DS (NJD_MATOFF+5)
```

Sets "diffuse" and "specular" for "Blend Alpha". The "ambient" setting is kept at the current value.

ChunkName : 'NJD_CM_AS'

(Chunk Material Ambient and Specular)

Outline:

Sets the "ambient" and "specular" values in "plist". The upper 8 bits of the "ambient" setting contain the alpha value. The upper 8 bits of the "specular" setting contain the exponent (0 - 16).

Format:

[headbits(13-8) | ChunkHead(7-0)][4(shortsize)][NRGB][ERGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_AS

NRGB:

NRGB8888(N?NOOP 255)

ERGB:

ERGB8888(E?Exponent 0?16)

Description:

#define NJD_CM_AS (NJD_MATOFF+6)

Sets "ambient" and "specular" for "Blend Alpha". The "diffuse" setting is kept at the current value.

ChunkName : 'NJD_CM_DAS'

(Chunk Material Diffuse Ambient and Specular)

Outline:

Sets the "diffuse" and "specular" values in "plist". The upper 8 bits of the "diffuse" setting contain the alpha value. The upper 8 bits of the "specular" setting contain the exponent (0 - 16).

Format:

[headbits(13-8) | ChunkHead(7-0)][6(shortsize)][ARGB][NRGB][ERGB]

headbits:

13-11 = SRC Alpha Instruction(3)

10- 8 = DST Alpha Instruction(3)

ChunkHead:

NJD_CM_DAS

ARGB:

ARGB8888

NRGB:

NRGB8888(N?NOOP 255)

ERGB:

ERGB8888(E?Exponent 0?16)

Description:

```
#define NJD_CM_DAS          (NJD_MATOFF+7)
Sets "diffuse, "ambient", and "specular" for "Blend Alpha".
```

3.8 Chunk Vertex

The Chunk Vertex gives the vertex list for the model. To allow the user to store desired data in the vertex list, it also uses the chunk format. By switching chunk types, the vertex, normal line, vertex color, user flag, and other items can be set as required in the vertex list. The upper 8 bits of the Chunk Head (headbits) are not used for the Chunk Vertex. If several chunk types were to be used simultaneously by the model, multiple library processing would be required. Currently, this is not supported. For one model, only one Chunk Vertex type is used. The header is based on Sint16, but the store array is as follows:

[ChunkHead(31-16) | bytesize(15-0)]

[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

<Format allowing high-speed processing with SH4>

NJD_CV_SH, NJD_CV_VN_SH

<Standard format without vertex normal line>

NJD_CV, NJD_CV_D8, NJD_CV_UF,

NJD_CV_NF, NJD_CV_S5, NJD_CV_S4, NJD_CV_IN

<Standard format with vertex normal line>

NJD_CV_VN, NJD_CV_VN_D8, NJD_CV_VN_UF,

NJD_CV_VN_NF, NJD_CV_VN_S5, NJD_CV_VN_S4,

NJD_CV_VN_IN

<32-bit vertex normal line with 10 bits each for x, y, z>

NJD_CV_VNX, NJD_CV_VNX_D8, NJD_CV_VNX_UF

The abbreviations have the following meanings.

VN	: use vertex normal
VNX	: 32bits vertex normal reserved(2) x(10) y(10) z(10)
SH	: SH4 optimize
D8	: Diffuse ARGB8888 only
S5	: Diffuse RGB565 and Specular RGB565
S4	: Diffuse RGB4444 and Specular RGB565
IN	: Diffuse(16) Specular(16)
NF	: NinjaFlags32 for extention
UF	: UserFlags32

The IndexOffset gives the start position for the library vertex intermediate buffer. For example, the vertex for the parent node with offset 0 is calculated, then the child is specified with an offset corresponding to the number of vertices in the parent node. When vertex processing data are stored starting at this position, the parent vertex processing results in the intermediate buffer will not be overwritten. By specifying index numbers in ascending order going towards the parent vertex, a polygon linking the parent and child vertices can be expressed.

"nbIndices" gives the number of vertices stored in the chunk.

ChunkName : 'NJD_CV_SH'

(Chunk Vertex for SH4 Optimize)

Outline:

Defines the vertex list in "vlist", without normal line. The data arrangement takes the matrix processing command characteristics of SH4 into consideration, to achieve high-speed processing.

Format:

[ChunkHead(31-16) | longsize(15-0)]

[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_SH

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,1.0F, ...

Description:

```
#define NJD_CV_SH (NJD_VERTOFF+0)
```

For matrix processing commands, the dummy 1.0F is inserted after x, y, z, to read data in 128-bit units. Because matrix processing is possible as is, high-speed execution can be realized. (Data to prove this effect are being compiled.) A normal line is not used. In models which do not perform light calculation and draw only the vertex color, no normal line is necessary. This chunk type should be used for such models.

ChunkName : 'NJD_CV_VN_SH'

(Chunk Vertex VertexNormal for SH4 Optimize)

Outline:

Defines the vertex list in "vlist", with normal line. The data arrangement takes the matrix processing command characteristics of SH4 into consideration, to achieve high-speed processing.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN_SH

longsize:

Offset until next chunk.

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,1.0F,nx,ny,nz,0.0F,...

Description:?

```
#define NJD_CV_VN_SH (NJD_VERTOFF+1)
```

For matrix processing commands, to read data in 128-bit units, the dummy 1.0F is inserted after x, y, z, and the dummy 0.0F after the normal line nx, ny, nz. Because matrix processing is possible as is, high-speed execution can be realized. (Data to prove this effect are being compiled.)

ChunkName : 'NJD_CV'

(Chunk Vertex)

Outline:

Defines the vertex list in "vlist", without normal line.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV

longsize:

Offset until next chunk.

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z, ...

Description:

```
#define NJD_CV (NJD_VERTOFF+2)
```

Gives the vertex list, without normal line.

ChunkName : 'NJD_CV_D8'

(Chunk Vertex Diffuse ARGB8888)

Outline:

Defines the vertex list in "vlist", with vertex color, without normal line.

Format:

[ChunkHead(31-16) | longsize(15-0)]

[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_D8

longsize:

Offset until next chunk.

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,D8888,...

Description:

```
#define NJD_CV_D8 (NJD_VERTOFF+3)
```

Gives the vertex list, with vertex color, without normal line. The vertex color is packed in Sint32 arrays.

ChunkName : 'NJD_CV_UF'

(Chunk Vertex UserFlag)

Outline:

Defines the vertex list in "vlist", without normal line. Provides a 32-bit user flag area.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_UF

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,UserFlags32, ...

Description:

```
#define NJD_CV_UF (NJD_VERTOFF+4)
```

Gives the vertex list, without normal line. Provides a 32- bit user flag area. Currently, the vertex color can be output to this area. In future releases, it will be possible to write user data to this area.

ChunkName : 'NJD_CV_NF'

(Chunk Vertex NinjaFlags32)

Outline:

Defines the vertex list in "vlist", without normal line. Provides a 32-bit Ninja expansion flag area.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_NF

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,NinjaFlags32, ...

Description:

```
#define NJD_CV_NF (NJD_VERTOFF+5)
```

Gives the vertex list, without normal line. Provides a 32- bit Ninja expansion flag area. This area is reserved for expanded Ninja functions.

ChunkName : 'NJD_CV_S5'

(Chunk Vertex Diffuse RGB565 and Specular RGB565)

Outline:

Defines the vertex list in "vlist", without normal line. Provides "diffuse" and "specular" vertex colors.

Format:

```
[ChunkHead(31-16) | longsize(15-0)]  
[IndexOffset(31-16) | nbIndices(15-0)][Data]
```

ChunkHead:

NJD_CV_S5

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,D565(31-16) | S565(15-0),...

Description:

```
#define NJD_CV_S5 (NJD_VERTOFF+6)
```

Gives the vertex list, without normal line. Vertex color can be set to "diffuse" and "specular". The "specular" setting is designed to enhance the color effect, but currently there is no setting method for the "specular" vertex color. Future releases will incorporate a method for setting "diffuse" and "specular" to be calculated for each vertex color by the converter, using the light source position of the scene (pre-light).

ChunkName : 'NJD_CV_S4'

(Chunk Vertex Specular RGB565 and Diffuse ARGB4444)

Outline:

Defines the vertex list in "vlist", without normal line. Provides "diffuse" and "specular" vertex colors.

Format:

```
[ChunkHead(31-16) | longsize(15-0)]  
[IndexOffset(31-16) | nbIndices(15-0)][Data]
```

ChunkHead:

```
NJD_CV_S4
```

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

```
x,y,z,D4444(31-16) | S565(15-0),...
```

Description:

```
#define NJD_CV_S4                (NJD_VERTOFF+7)
```

Gives the vertex list, without normal line. Vertex color can be set to "diffuse" and "specular". The "specular" setting is designed to enhance the color effect. "diffuse" can be set to the alpha value ARGB4444. Currently there is no setting method for the "specular" vertex color. Future releases will incorporate a method for setting "diffuse" and "specular" to be calculated for each vertex color by the converter, using the light source position of the scene (pre-light).

ChunkName : 'NJD_CV_IN'

(Chunk Vertex INtensity Diffuse and Specular)

Outline:

Defines the vertex list in "vlist", without normal line. The high-speed Intensity mode is used to provide the vertex color. "diffuse" and "specular" vertex colors are available.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_IN

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,D16 | S16,...

Description:

```
#define NJD_CV_IN (NJD_VERTOFF+8)
```

Gives the vertex list, without normal line. The Intensity mode is used to provide the vertex color. The "specular" setting is designed to enhance the color effect. In Intensity mode, "diffuse" and "specular" are specified only by the intensity. Both values have a 16-bit range and are set in D16 and S16. Currently there is no setting method for the "specular" vertex color. Future releases will incorporate a method for setting "diffuse" and "specular" to be calculated for each vertex color by the converter, using the light source position of the scene (pre-light).

ChunkName : 'NJD_CV_VN'

(Chunk Vertex VertexNormal)

Outline:

Defines the vertex list in "vlist", with normal line.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz, ...

Description:

```
#define NJD_CV_VN (NJD_VERTOFF+9)
```

Gives the vertex list, with normal line. This is the most commonly used vertex list.

ChunkName : 'NJD_CV_VN_D8'

(Chunk Vertex VertexNormal and Diffuse ARGB8888)

Outline:?

Defines the vertex list in "vlist", with vertex color, with normal line.

Format:?

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN_D8

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz,D8888,...

Description:

```
#define NJD_CV_VN_D (NJD_VERTOFF+10)
```

Gives the vertex list, with vertex color, with normal line. The vertex color is packed in Sint32 arrays.

ChunkName : 'NJD_CV_VN_UF'

(Chunk Vertex VertexNormal and UserFlags32)

Outline:

Defines the vertex list in "vlist", with normal line. Provides a 32-bit user flag area.

Format:

[ChunkHead(31-16) | longsize(15-0)]

[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN_UF

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz,UserFlags32,...

Description:

```
#define NJD_CV_VN_UF (NJD_VERTOFF+11)
```

Gives the vertex list, with normal line. Provides a 32-bit user flag area. Currently, the vertex color can be output to this area. In future releases, it will be possible to write user data to this area.

ChunkName : 'NJD_CV_VN_NF'

(Chunk Vertex VertexNormal and NinjaFlags32)

Outline:

Defines the vertex list in "vlist", with normal line. Provides a 32-bit Ninja expansion flag area.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN_NF

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz,NinjaFlags32,...

Description:

```
#define NJD_CV_VN_NF (NJD_VERTOFF+12)
```

Gives the vertex list, with normal line. Provides a 32-bit Ninja expansion flag area. This area is reserved for expanded Ninja functions.

ChunkName : 'NJD_CV_VN_S5'

(Chunk Vertex VertexNormal, Diffuse RGB565 and Specular RGB565)

Outline:

Defines the vertex list in "vlist", with normal line. Provides "diffuse" and "specular" vertex colors.

Format:?

[ChunkHead(31-16) | bytesize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN_S5

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz,D565(31-16) | S565(15-0),...

Description:

```
#define NJD_CV_VN_S5 (NJD_VERTOFF+13)
```

Gives the vertex list, with normal line. Vertex color is packed in Sint32 arrays. Vertex color can be set to "diffuse" and "specular". The "specular" setting is designed to enhance the color effect, but currently there is no setting method for the "specular" vertex color. Future releases will incorporate a method for setting "diffuse" and "specular" to be calculated for each vertex color by the converter, using the light source position of the scene (pre-light).

ChunkName : 'NJD_CV_VN_S4'

(Chunk Vertex VertexNormal, Specular RGB565 and Diffuse ARGB4444)

Outline:

Defines the vertex list in "vlist", with normal line. Provides "diffuse" and "specular" vertex colors.

Format:

```
[ChunkHead(31-16) | longsize(15-0)]  
[IndexOffset(31-16) | nbIndices(15-0)][Data]
```

ChunkHead:

NJD_CV_VN_S4

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz,D4444(31-16) | S565(15-0),...

Description:

```
#define NJD_CV_VN_S4 (NJD_VERTOFF+14)
```

Gives the vertex list, with normal line. Vertex color is packed in Sint32 arrays. Vertex color can be set to "diffuse" and "specular". The "specular" setting is designed to enhance the color effect. "diffuse" can be set to the alpha value ARGB4444. Currently there is no setting method for the "specular" vertex color. Future releases will incorporate a method for setting "diffuse" and "specular" to be calculated for each vertex color by the converter, using the light source position of the scene (pre-light).

ChunkName : 'NJD_CV_VN_IN'

(Chunk Vertex VertexNormal, INtensity Diffuse and Specular)

Outline:

Defines the vertex list in "vlist", with normal line. The high-speed Intensity mode is used to provide the vertex color. "diffuse" and "specular" vertex colors are available.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VN_IN

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nx,ny,nz,D16 | S16,...

Description:

```
#define NJD_CV_VN_IN (NJD_VERTOFF+15)
```

Gives the vertex list, with normal line. Vertex color is packed in Sint32 arrays. The Intensity mode is used to provide the vertex color. The "specular" setting is designed to enhance the color effect. In Intensity mode, "diffuse" and "specular" are specified only by the intensity. Both values have a 16-bit range and are set in D16 and S16. Currently there is no setting method for the "specular" vertex color. Future releases will incorporate a method for setting "diffuse" and "specular" to be calculated for each vertex color by the converter, using the light source position of the scene (pre-light).

ChunkName : 'NJD_CV_VNX'

(Chunk Vertex VertexNormal 32bits(X))

Outline:

Defines the vertex list in "vlist", with 32-bit normal line.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VNX

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nxyz32, ...

Description:

```
#define NJD_CV_VNX (NJD_VERTOFF+16)
```

Gives the vertex list, with 32-bit normal line. Vertex normal line data are reduced, in order to decrease the data amount. x,y,z are assigned 10 bits each, and the remaining 2 bits are reserved. Resolution is 1024. Using the vertex normal line at this resolution, glow processing is performed.

ChunkName : 'NJD_CV_VNX_D8'

(Chunk Vertex VertexNormal 32bits(X) and Diffuse ARGB8888)

Outline:

Defines the vertex list in "vlist", with 32-bit normal line, and with vertex color.

Format:

[ChunkHead(31-16) | longsize(15-0)]
[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VNX_D8

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nxyz32,D8888,...

Description:

```
#define NJD_CV_VNX_D8 (NJD_VERTOFF+17)
```

Gives the vertex list, with 32-bit normal line. Vertex normal line data are reduced, in order to decrease the data amount. x,y,z are assigned 10 bits each, and the remaining 2 bits are reserved. Resolution is 1024. Using the vertex normal line at this resolution, glow processing is performed. Vertex color is provided, packed in Sint32 arrays.

ChunkName : 'NJD_CV_VNX_UF'

(Chunk Vertex VertexNormal 32bits(X) and UserFlags32)

Outline:

Defines the vertex list in "vlist", with 32-bit normal line. Provides a 32-bit user flag area.

Format:

[ChunkHead(31-16) | longsize(15-0)]

[IndexOffset(31-16) | nbIndices(15-0)][Data]

ChunkHead:

NJD_CV_VNX_UF

longsize:

Offset until next chunk..

IndexOffset:

Gives buffer start position for vertex intermediate buffer.

nbIndices:

Gives number of vertices.

Data:

x,y,z,nxyz32,UserFlags32,...

Description:

```
#define NJD_CV_VNX_UF (NJD_VERTOFF+18)
```

Gives the vertex list, with 32-bit normal line. Vertex normal line data are reduced, in order to decrease the data amount. x,y,z are assigned 10 bits each, and the remaining 2 bits are reserved. Resolution is 1024. Using the vertex normal line at this resolution, glow processing is performed. Vertex color is provided, packed in Sint32 arrays. Provides a 32-bit user flag area.

3.9 Chunk Volume

Chunk Volume is provided only as collision and modifier volume. It cannot be used directly by the library for drawing, and it has no material data. Currently, there are three Chunk Volume types. `NJD_CO_P3` consists of separate triangular data, and `NJD_CO_P4` of separate quadrilateral data. In 3D Studio MAX, only triangular data are output, but NinjaExport has an option for connecting quadrilateral shapes with a screen angle of max. 0.1 degrees and restoring separate quadrilateral shapes for output. This allows the creation of quadrilateral collisions also in 3D Studio MAX. When the original data are for mixed triangular/quadrilateral/N-sided polygons, a converter option can divide them into triangular polygon data and recreate separate triangular and quadrilateral data. In this case, N-sided data will be eliminated, and the "plist" contains `NJD_CO_P3` and `NJD_CO_P4`. `NJD_CO_ST` is the triangular strip Chunk Volume. Chunk Volume can contain a Chunk Strip area (see below) and user flag areas (16, 32, 48 bits) for equivalent polygons. The material color set with the modeler can be output to the user flag area for each polygon.

The triangular Chunk Volume `NJD_CO_P3` can be used by the modifier volume as is. However, the modifier volume must be a closed 3D space. This is a hardware specification. Note that the modifier volume can only use separate triangular data. The upper 8 bits of the Chunk Head (headbits) are not used for the Chunk Volume.

[ChunkHead(15-0)][shortsize(15-0)]

[UserOffset(15-14) | nbPolygon(13-0)][Data]

The user flag area is handled in 16-bit units (because "plist" is a Sint16 primary array). Its size can be 16, 32, or 48 bits. It is set with the UserOffset allocated to the top 2 bits of nbPolygon which gives the number of polygons.

UserFlags Offset:

```
#define NJD_UFO_SHIFT      14
#define NJD_UFO_0          (0<<NJD_UFO_SHIFT)
#define NJD_UFO_1          (1<<NJD_UFO_SHIFT)
#define NJD_UFO_2          (2<<NJD_UFO_SHIFT)
#define NJD_UFO_3          (3<<NJD_UFO_SHIFT)
#define NJD_UFO_MASK       (0x3<<NJD_UFO_SHIFT) /* 0 - 3 */
NJD_UFO_0: UserFlags size 0
NJD_UFO_1: UserFlags size 16 bits
NJD_UFO_2: UserFlags size 32 bits
NJD_UFO_3: UserFlags size 48 bits
```

ChunkName : 'NJD_CO_P3'

(Chunk vOlume Polygon3)

Outline:

Defines the volume polygon list in "plist". This is not used for direct drawing, but for collision and modifier volumes. Material information is not included, but there is a user flag area to which the polygon color can be output.

Format:

```
[ChunkHead(15-0)][shortsize(15-0)]  
[UserOffset(15-14) | nbPolygon(13-0)][Data]
```

ChunkHead:

```
NJD_CO_P3
```

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbPolygon:

Gives number of polygons.

Data:

```
index0, index1, index2, UserflagPoly0(*N),  
index3, index4, index5, UserflagPoly1(*N), ...
```

Description:

```
#define NJD_CO_P3 (NJD_VOLOFF+0)
```

These are separate triangular data for collision and modifier volumes. Material information is not included, but polygon color can be output to a user flag area. The polygon color is as set for the material. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library simply skips the user flag area without doing anything. When original data are mixed triangular/quadrilateral/N-sided, a converter option can divide them into triangular data and output these to NJD_CO_P3.

ChunkName : 'NJD_CO_P4'

(Chunk vOlume Polygon4)

Outline:

Defines the volume polygon list in "plist". This is not used for direct drawing, but for collision and modifier volumes. Material information is not included, but there is a user flag area to which the polygon color can be output.

Format:

```
[ChunkHead(15-0)][shortsize(15-0)]  
[UserOffset(15-14) | nbPolygon(13-0)][Data]
```

ChunkHead:

```
NJD_CO_P4
```

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbPolygon:

Gives number of polygons.

Data:

index0, index1, index2, index3, UserflagPoly0(*N),
index4, index5, index6, index7, UserflagPoly1(*N), ...

Description:

```
#define NJD_CO_P4 (NJD_VOLOFF+1)
```

These are separate quadrilateral data for collision and modifier volumes. Material information is not included, but polygon color can be output to a user flag area. The polygon color is as set for the material. In 3D Studio MAX, the object color can be used. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library simply skips the user flag area without doing anything. In 3D Studio MAX, only triangular data are output, but a converter option allows connecting two quadrilateral shapes with a screen angle of max. 0.1 degrees and restoring separate quadrilateral shapes for output. This allows the creation of separate quadrilateral data.

ChunkName : 'NJD_CO_ST'

(Chunk vOluMe Triangle STrip)

Outline:

Defines the volume polygon list in "plist". This is not used for direct drawing, but for collision and modifier volumes. Material information is not included, but there is a user flag area to which the polygon color can be output.

Format:

```
[ChunkHead(15-0)][shortsize(15-0)]
[UserOffset(15-14) | nbPolygon(13-0)][Data]
```

ChunkHead:

```
NJD_CO_ST
```

shortsize:

Offset until next chunk..

UserOffset:

Gives user flag area size.

nbPolygon:

Gives number of polygons.

Data:

```
[flag(15) | len(14-0), i0, i1, i2, Userflag2(*N), i3, Userflag3(*N), ...]
```

Description:

```
#define NJD_CO_ST (NJD_VOLOFF+2)
```

Used for collision. Compared to NJD_CO_P3 and NJD_CO_P4, the collision data size is reduced. However, note that triangular strip connection proceeds in the most effective direction which is not necessarily the direction intended by the user. The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "i?" is the polygon vertex index. Material information is not included, but polygon color can be output to a user flag area. The polygon color is as set for the material. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library simply skips the user flag area without doing anything. When original data are mixed triangular/quadrilateral/N-sided polygon data, the converter automatically divides all into separate triangular data and outputs these to NJD_CO_ST.

3.10 Chunk Strip

Chunk Strip creates the triangular strip, using the entry number in the vertex intermediate buffer created in the library from the Chunk Vertex list. It can include a vertex color, vertex normal line, and individual user flag areas for each polygon. Because the polygon side includes vertex color, individual vertex color setting for each polygon is possible also at the same vertex. By using the normal line for the polygon, the edge between polygons can be raised. Discontinuity of "softimage" is supported, and the "softimage" vertex normal line can be output as is. Polygon color set for the material can be output to the user flag area. It is not possible to have vertex color both in Chunk Vertex and Chunk Strip. If vertex color output has been specified on the Chunk Strip side, vertex color data cannot be output on the Chunk Vertex side. It is also not possible to have a normal line both in Chunk Vertex and Chunk Strip. If normal line output has been specified on the Chunk Strip side, normal line data cannot be output on the Chunk Vertex side.

The upper 8 bits of the Chunk Head (headbits) are used for the attribute flags (ChunkFlags) set for the material.

The abbreviations have the following meanings.

IL :	Ignore light
IS :	Ignore specular
IA :	Ignore ambient
UA :	Use alpha
DB :	Double side
FL :	Flat shading
ENV :	Environment mapping

Flag STRip:

```
#define NJD_FST_SHIFT      8
#define NJD_FST_IL         (0x01<<NJD_FST_SHIFT)
#define NJD_FST_IS         (0x02<<NJD_FST_SHIFT)
#define NJD_FST_IA         (0x04<<NJD_FST_SHIFT)
#define NJD_FST_UA         (0x08<<NJD_FST_SHIFT)
#define NJD_FST_DB         (0x10<<NJD_FST_SHIFT)
#define NJD_FST_FL         (0x20<<NJD_FST_SHIFT)
#define NJD_FST_ENV        (0x40<<NJD_FST_SHIFT)
#define NJD_FST_MASK       (0xFF<<NJD_FST_SHIFT)
```

The Chunk Strip format is indicated below. Note that UserFlags are inserted in polygon units after index2. This is because the first triangular polygon is created at the third point. From the 4th point and later, a triangular polygon is created at every point. Each point is followed by a user flag.

No polygon vertex normal line, no vertex color, no texture:

[ChunkFlags(15-8) | ChunkHead(7-0)]

[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)]

[flag(15) | len(14-0), index0(15-0),
 index1(15-0),
 index2, UserFlag2(*N), ...]

No polygon vertex normal line, no vertex color, with texture:

```
[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)]
[flag(15) | len(14-0),      index0(15-0), U0(15-0), V0(15-0),
      index1, U1, V1,
      index2, U2, V2, UserFlag2(*N), ... ]
```

With polygon vertex normal line, no vertex color, no texture:

```
[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)]
[flag(15) | len(14-0),      index0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
      index1, vnx1, vny1,
      index2, vnx2, vny2, vnz2, UserFlag2(*N),
      index3, vnx2, vny2, vnz2, UserFlag3(*N), ... ]
```

With polygon vertex normal line, no vertex color, with texture:

```
[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)]
[flag(15) | len(14-0),
      index0(15-0), U0(15-0), V0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
      index1, U1, V1, vnx1, vny1,
      index2, U2, V2, vnx2, vny2, vnz2, UserFlag2(*N),
      index3, U3, V3, vnx3, vny3, vnz3, UserFlag3(*N), ... ]
```

No polygon vertex normal line, with vertex color, no texture:

```
[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)]
[flag(15) | len(14-0),
      index0(15-0), AR0(15-0), GB0(15-0),
      index1, AR1, GB1,
      index2, AR2, GB2, UserFlag2(*N),
      index3, AR3, GB3, UserFlag3(*N), ... ]
```

No polygon vertex normal line, with vertex color, with texture:

```
[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)]
[flag(15) | len(14-0),
      index0(16), U0(16), V0(16), AR0(16), GB0(16),
      index1, U1, V1, AR1, GB1,
      index2, U2, V2, AR2, GB2, UserFlag2(*N), ... ]
```

The following two types of UV value expressions are available: 0-255 UVN, and high-resolution 0-1023 UVH. UVN is a conventional expression which has been used in Basic a Model. However, resolution suffers at sizes exceeding 256. With UVH and high-resolution mode, 1024 x 1024 texture can be specified in 1-pixel units. But compared to UVN, the texture repeat count of UVH decreases proportionally to the increase in resolution (32 times for UVH vs. 128 times for UVN). UVN and UVH can be switched by convert option for the whole of model tree and also material names can be used to switch at each model unit. In case that the UV value is specified by material name, UV value expressions are changed by setting to only one material (among some materials used for single model). The default is UVN.

UVN : Normal type Uv (0-255)

UVH : Hiresolution type Uv (0-1023)

In the same way as for Chunk Volume, the user flag area in Chunk Strip is allocated by the top 2 bits of nbStrip (UserOffset). It is handled in 16-bit units (because "plsit" is a Sint16 primary array). Its size can be 16, 32, or 48 bits.

UserFlags Offset:

```
#define NJD_UFO_SHIFT      14
#define NJD_UFO_0          (0<<NJD_UFO_SHIFT)
#define NJD_UFO_1          (1<<NJD_UFO_SHIFT)
#define NJD_UFO_2          (2<<NJD_UFO_SHIFT)
#define NJD_UFO_3          (3<<NJD_UFO_SHIFT)
#define NJD_UFO_MASK       (0x3<<NJD_UFO_SHIFT) /* 0 - 3 */
NJD_UFO_0: UserFlags size 0
NJD_UFO_1: UserFlags size 16 bits
NJD_UFO_2: UserFlags size 32 bits
NJD_UFO_3: UserFlags size 48 bits
```

ChunkName : 'NJD_CS'

(Chunk Strip)

Outline:

Defines the polygon list in "plist", without polygon vertex normal line, without vertex color, without texture.

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0), index0(15-0),
index1(15-0),
index2, UserFlag2(*N), ...]

Description:

```
#define NJD_CS (NJD_STRIPOFF+0)
```

The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_UVN'

(Chunk Strip UVN)

Outline:

Defines the polygon list in "plist", without polygon vertex normal line, without vertex color, with texture. The UV value is given by UVN (0 - 255).

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_UVN

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0), index0(15-0), U0(15-0), V0(15-0),
index1, U1, V1,
index2, U2, V2, UserFlag2(*N), ...]

Description:

```
#define NJD_CS_UVN (NJD_STRIPOFF+1)
```

The UV value is given by UVN (0 - 255). The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_UVH'

(Chunk Strip UVH)

Outline:

Defines the polygon list in "plist", without polygon vertex normal line, without vertex color, with texture. The UV value is given by UVH (0 - 1023).

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_UVH

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0), index0(15-0), U0(15-0), V0(15-0),
index1, U1, V1,
index2, U2, V2, UserFlag2(*N), ...]

Description:

```
#define NJD_CS_UVH (NJD_STRIPOFF+2)
```

The UV value is given by UVH (0 - 1023). The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_VN'

(Chunk Strip VertexNormal)

Outline:

Defines the polygon list in "plist", with polygon vertex normal line, without vertex color, without texture.

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_VN

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0), index0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
index1, vnx1, vny1, vny1,
index2, vnx2, vny2, vnz2, UserFlag2(*N),
index3, vnx2, vny2, vnz2, UserFlag3(*N), ...]

Description:

```
#define NJD_CS_VN (NJD_STRIPOFF+3)
```

The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_UVN_VN'

(Chunk Strip UVN VertexNormal)

Outline:

Defines the polygon list in "plist", with polygon vertex normal line, without vertex color, with texture. The UV value is given by UVN (0 - 255).

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_UVN_VN

shortsize:

Offset until next chunk.. Based on IFF format.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0),
index0(15-0), U0(15-0), V0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
index1, U1, V1, vnx1, vny1, vny1,
index2, U2, V2, vnx2, vny2, vnz2, UserFlag2(*N),
index3, U3, V3, vnx3, vny3, vnz3, UserFlag3(*N), ...]

Description:

```
#define NJD_CS_UVN_VN          (NJD_STRIPOFF+4)
```

The UV value is given by UVN (0 - 255). The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_UVH_VN'

(Chunk Strip UVH VertexNormal)

Outline:

Defines the polygon list in "plist", with polygon vertex normal line, without vertex color, with texture. The UV value is given by UVH (0 - 1023).

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_UVH_VN

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0),
index0(15-0), U0(15-0), V0(15-0), vnx0(15-0), vny0(15-0), vnz0(15-0),
index1, U1, V1, vnx1, vny1, vnz1,
index2, U2, V2, vnx2, vny2, vnz2, UserFlag2(*N),
index3, U3, V3, vnx3, vny3, vnz3, UserFlag3(*N), ...]

Description:

```
#define NJD_CS_UVH_VN          (NJD_STRIPOFF+5)
```

The UV value is given by UVH (0 - 1023). Note that the maximum repeat value possible with UVH (32 repetitions) is lower than for UVN. The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_D8'

(Chunk Strip Diffuse ARGB8888)

Outline:

Defines the polygon list in "plist", without polygon vertex normal line, with vertex color, without texture.

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_D8

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[flag(15) | len(14-0),
index0(15-0), AR0(15-0), GB0(15-0),
index1, AR1, GB1,
index2, AR2, GB2, UserFlag2(*N),
index3, AR3, GB3, UserFlag3(*N), ...]

Description:

```
#define NJD_CS_D8          (NJD_STRIPOFF+7)
```

The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

ChunkName : 'NJD_CS_UVH_D8'

(Chunk Strip UVH Diffuse ARGB8888)

Outline:

Defines the polygon list in "plist", without polygon vertex normal line, with vertex color, with texture. The UV value is given by UVH (0 - 1023).

Format:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[shortsize(15-0)][UserOffset(15-14) | nbStrip(13-0)][Data]

ChunkFlags:

NJD_FST_IL(Ignore light source), NJD_FST_IS(Ignore specular),
NJD_FST_IA(Ignore ambient), NJD_FST_UA?Use Alpha?,
NJD_FST_DB(Dual-sided), NJD_FST_FL(Flat shading),
NJD_FST_ENV(Environment mapping)

ChunkHead:

NJD_CS_UVH_D8

shortsize:

Offset until next chunk.

UserOffset:

Gives user flag area size.

nbStrip:

Gives number of strip vertices.

Data:

[ChunkFlags(15-8) | ChunkHead(7-0)]
[byteSize(15-0)][UserOffset(15-14) | nbStrip(13-0)]
[flag(15) | len(14-0),
index0(16), U0(16), V0(16), AR0(16), GB0(16),
index1, U1, V1, AR1, GB1,
index2, U2, V2, AR2, GB2, UserFlag2(*N), ...]

Description:

```
#define NJD_CS_UVH_D8      (NJD_STRIPOFF+8)
```

The UV value is given by UVH (0 - 1023). Note that the maximum repeat value possible with UVH (32 repetitions) is lower than for UVN. The flag specifies the triangular rotation direction (right rotation/left rotation) at the strip start. The Chunk Model can switch between left and right rotation using a negative or positive prefix. Negative prefix means right rotation. "len" indicates the number of vertices included in the strip. "index?" is the polygon vertex index. There is a user flag area after the polygon index. The size of this area is determined by the UserOffset value (NJD_UFO_0: none; NJD_UFO_1: 16 bits; NJD_UFO_2: 32 bits; NJD_UFO_3: 48 bits). The library skips the user flag area without doing anything. Mixed triangular/quadrilateral/N-sided polygon data are automatically divided into separate triangular data and converted to strips for output.

4 ASCII Output Precautions

This section contains precautions for Chunk Model .nja file output.

Because "vlist" uses a Sint32 array format, float values for vertex or vertex normal lines cannot be input as is. Therefore ASCII output is expressed in hexadecimal notation. When wishing to have a value recognized as a float value, use the converter option to output the float value as a comment.

In the Chunk Model, all flags are also output as character strings. Character strings were chosen to be non-common and as short as possible. The character string appears unchanged in the .nja file, with the "NJD_" part removed. For details, refer to the section on `NjDef.h`.



10. Nindows Tutorial

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.

1.1 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

0.Preparing the Ninja application.

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

1.Include the Nindows header file

Add the following line to the source file.

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

2.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()`.

* Assign the value that is added 3 to the number used in the application, as three textures are used for the font of Nindows.

3.Call the function to execute Nindows

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

```
return nwExecute();
```

4. Call the function to exit Nindows

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

```
nwExitSystem();
```

5.Linkng 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.

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

Table 2.1 *List of functions used in integrating Nindows*

nwInitSystem	Initialization function
Format	<code>void nwInitSystem(UINT32 numTextures)</code>
Parameters	NumTextures Number of texture memory lists
Return value	None Function
	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()
Note	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

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

Initialization function

nwExecute

Format	<code>Sint32 nwExecute(void)</code>
Parameters	None
Return value	If 'Exit' is selected from the System Menu it returns all other cases it returns <code>NJD_USER_CONTINUE</code> .
Function	Performs all Nindows drawing.
Reference	<code>njUserMain()</code>
Note	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 <code>njUserMain()</code> , the application can be exited by selecting the 'Exit' menu.
Example	<pre>Sint32 njUserMain(void) { : : : return nwExecute(); }</pre>

Execution Function

nwInitResource

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

Initialization Function

3 Using Nindows and Nindows Utilities

3.1 Using Nindows

In section 2, 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.

Table 2.2 List of System Menu items

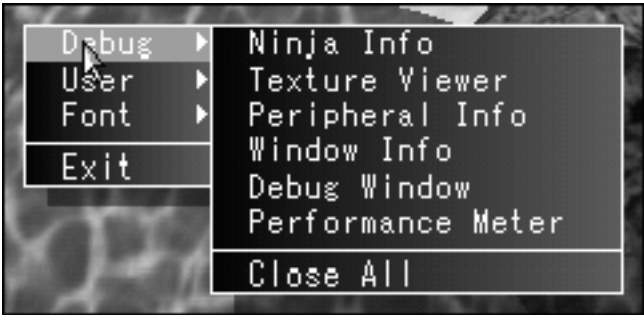


Figure 1.1 In this diagram, The System Menu is displayed with the right button and the "Debug" menu is selected

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.

Table 2.3 *List of Nindows Utilities*

Ninja Info Window



Figure 1.2 *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

Table 2.4 Information shown in the Ninja Info Window

Texture Viewer Window

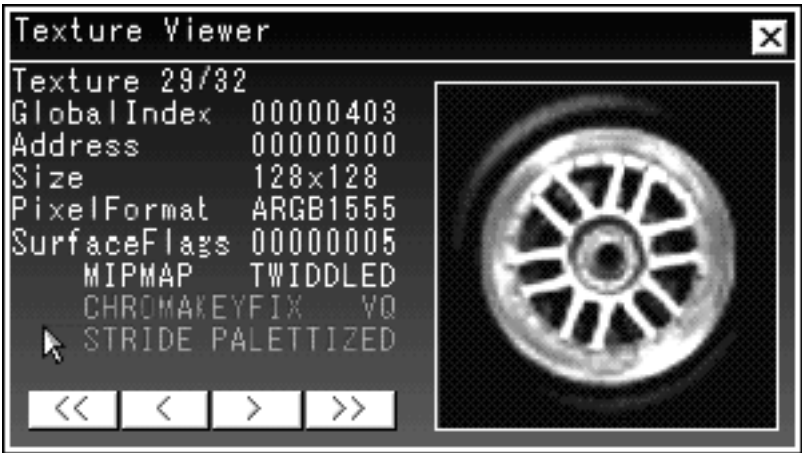


Figure 1.3 Texture Viewer Window

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.

Table 2.5 Texture Viewer Window Information

Peripheral Info Window



Figure 1.4 Peripheral Info Window

Window which 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

Table 2.6 Info Displayed in the Peripheral Info Window

Window Info Window

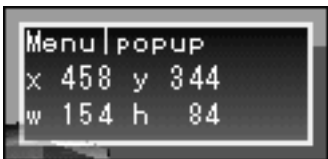


Figure 1.5 Peripheral Info Window

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

Table 2.7 *Information Displayed in the Peripheral Info Window*

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

Debugging Window



Figure 1.6 *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

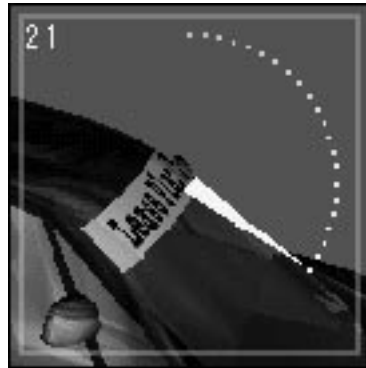


Figure 1.7 *Performance Meter Window*

Provides an intuitive understanding of an application's performance (calculation, drawing speed).

If the meter revolves once per second, the frame rate is 60fps.

3.3 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.

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

Table 2.8 *Types of Windows and Window Classes*

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()`.

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.)

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.

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.

Furthermore, scrollbars (class `NWD_WC_SCROLLBAR`), 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 `NWHWND` 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

Table 2.9 *List of Functions for Creating Windows*

Nindows API**nwCreateWindow****Window Creation Functions**

Format	<code>NHWND nwCreateWindow(Sint32 wClass, Sint8* caption, Sint32</code>
style,	
<code>hWndParent)</code>	<code>Sint32 x, Sint32 y, Sint32 w, Sint32 h, NHWND</code>
Parameters	<code>wClass</code> Window class <code>caption</code> Window name (caption) <code>style</code> Window style <code>x,y</code> Upper left coordinate of the client area <code>w,h</code> Width and height of the client area <code>hWndParent</code> 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	<code>nwDestroyWindow()</code> , <code>nwCreateMenuWindow()</code> , <code>nwCreateEditWindow()</code> , <code>nwCreateScrollBar()</code> , <code>nwCreateButton()</code> NWS_WIN structure
Note	For creating menu windows, edit windows, scrollbars, and buttons
it is	recommended to use the more convenient <code>nwCreateMenuWindow()</code> , <code>nwCreateEditWindow()</code> , <code>nwCreateScrollBarArray()</code> , <code>nwCreateButton()</code> .
Example	<code>// Creates a window</code> <code>NHWND hWnd;</code> <code>hWnd = nwCreateWindow(NWD_WC_WIN,</code> <code> "Test Window",</code> <code> NWD_WS_CAPTION NWD_WS_BORDER NWD_WS_SHADOW,</code> <code> 50, 50, 100, 100,</code> <code> NULL);</code>

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
NWD_WS_SB_HORZ	Create a horizontal scrollbar
NWD_WS_SB_VERT	Create a vertical scrollbar

nwDestroyWindow

Window Creation Function

Format `void nwDestroyWindow(NWHWND hWnd)`

Parameters `hWnd` the handle of the window to be destroyed

Return value `None`

Function `Destroys the window`

Reference `NwCreateWindow(), NWS_WIN structure`

Note `If a callback function is set in hWnd->desructor, it calls back that function.`

Example

```
// Destroys a window
nwDestroyWindow(hWnd);
```

Callback Functions**ClientDrawCallback****Window Callback Function**

Format	Void ClientDrawCallback(NHWND hWnd)	
Parameters	HWnd	Handle of the window where the callback originated
Return value	None	
Function	An application defined function which a window calls back for drawing	
Reference	nwCreateWindow(),NWS_WIN 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		

4.6 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.

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

Table 2.10 *List of Functions for Creating Windows*

Windows API

nwFindWindow

Window Support Function

Format	NHWND nwFindWindow(NHWND hWnd, Sint8* caption)
Parameters	<div>HWnd The parent window which starts searching for the window</div> <div>Caption The caption string of the window being searched for</div>
Return value	If the window is found, it returns its window handle else it returns NULL.
Function	<div>Searches the specified parent window's child windows for the window with the specified caption string</div> <div>If you want to search all windows, specify NULL for the parent window.</div>
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	<div>x, y Screen coordinates</div>
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	<div>hWnd Window handle</div> <div>rect Address which holds the rectangle information</div>
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	<pre>NWS_RECT rect; nwGetClientRect(hWnd, &rect);</pre>

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	If NWD_WS_OFFSET is specified in the window style, the coordinates are relative to the parent window.
Example	Sint32 x, y; nwGetWindowPos(hWnd, &x, &y);

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	NWS_RECT rect; nwGetWindowRect(hWnd, &rect);

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 nwGetWindowStyle(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

```
// Gets the window hWnd's in buf
Sint8 buf[256];
nwGetWindowText(hWnd, buf, sizeof(buf));
```

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

Example

```
If NWD_WS_SHADING is not specified in the window style, the color of
the upper left vertex is applied to all vertices.
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);
```


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	// Moves the window to the point (100, 50). nwSetWindowPos(hWnd, 100, 50);

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	// Changes the width and height of the window's client area to (128, 64) nwSetWindowSize(hWnd, 128, 64);

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	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	// Removes the shadow from the window and attached a caption nwSetWindowStyle(hWnd, ~NWD_WS_SHADOW, NWD_WS_CAPTION);

NwSetWindowText

Format	Bool nwSetWindowStyle(NWHWND hWnd, Sint8* caption)
Parameters	<div>HWnd Window handle</div> <div>caption Pointer to the NULL terminal caption string</div>
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**

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

Window Support Function**Structure**

	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	typedef struct _NWS_RGBA { Uint8 r; Uint8 g; Uint8 b; Uint8 a; } NWS_RGBA;	
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	typedef struct _NWS_RECT { Sint32 left; Sint32 top; Sint32 right; Sint32 bottom; } NWS_RECT;	
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.

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
<code>nwScrWinEnableScroll</code>	Enables or disables scrolling in the scroll window
<code>nwScrWinSetClip</code>	Sets the scrolling area of the scroll window
<code>nwScrWinScroll</code>	Scrolls the scroll window

Table 2.11 *List of Functions for Creating scroll windows*

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	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	This function's settings are only valid for scrolling with the mouse. Scrolling via nwScrWinScroll() is always valid in every direction. Example// Enables scrolling in every direction nwScrWinEnableScroll(hWnd, NWD_ES_VERTICAL NWD_ES_HORIZONTAL);

nwScrWinSetClip**Scroll Window Function**

Format	Bool nwScrWinSetClip(NWHWND 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	Sets the range of scrolling in the client area of the scroll window.
Reference	NwScrWinEnableScroll(),nwScrWinScroll()
Note	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	// Sets the range of scrolling to (-10,-10)-(10,10) NWS_RECT rect = {-10, -10, 10, 10}; NwScrWinSetClip(hWnd, &rect);

nwScrWinScroll**Scroll Window Function**

Format	Bool nwScrWinScroll(NWHWND 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 dots	Scrolls the client area of the scroll window by the specified number of
Reference	NwScrWinEnableScroll(),nwScrWinSetClip()
Note	
Example	<pre>//Scrolls up one dot at a time Sint32 njUserMain(void) { NWHWND hWnd = nwFindWindow(NULL, "Scroll Window"); if (hWnd) { nwScrWinScroll(hWnd, 0, 1); } : : return nwExecute(); }</pre>

nwScrWinGetScroll**Scroll Window Function**

Format	Bool nwScrWinGetScroll(NWHWND hWnd, Sint32* x, Sint32* y)
Parameters	hWnd Window handle of the scroll window x, y Pointer 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	<pre>//Sets scrolling Sint32 x, y; nwScrWinGetScroll(hWnd, &x, &y); nwScrWinSetScroll(hWnd, -x, -y);</pre>

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_SCROLLWIN`).

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()`.

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

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

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

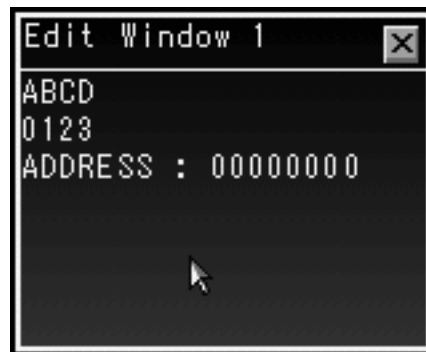


Figure 1.8 An example of creating an Edit Window

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()`.

*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

Table 2.12 *List of Functions for Creating Edit Windows*

Nindows API

nwCreateEditWindow

Window Creation Function

Format	<code>NHWND nwCreateEditWindow(Sint32 lines, Sint8* caption, Sint32 style, Sint32 x, Sint32 y, Sint32 w, Sint32 h, NHWND hWndParent)</code>	
Parameters	<code>lines</code>	Maximum number of text lines
	<code>caption</code>	Window name string (caption)
	<code>style</code>	Window style
	<code>x,y</code>	Upper left coordinate of the client area
	<code>w,h</code>	Width and height of the client area
	<code>hWndParent</code>	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	<code>nwDestroyWindow()</code> , <code>nwCreateMenuWindow()</code> , <code>nwCreateEditWindow()</code> , <code>nwCreateScrollBar()</code> , <code>nwCreateButton()</code> <code>NWS_WIN</code> 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	<code>Bool nwEditWinAddString(NHWND hWnd, Sint8* string)</code>	
Parameters	<code>hWnd</code>	Window handle of the edit window that text is being added to
	<code>string</code>	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	<code>nwCreateEditWindow()</code> , <code>nwEditWinPrintf()</code>	
Note		
Example	<code>nwEditWinAddString(hWnd, "AddText\n");</code>	

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 fmt printf() format text string
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. 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	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.

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.

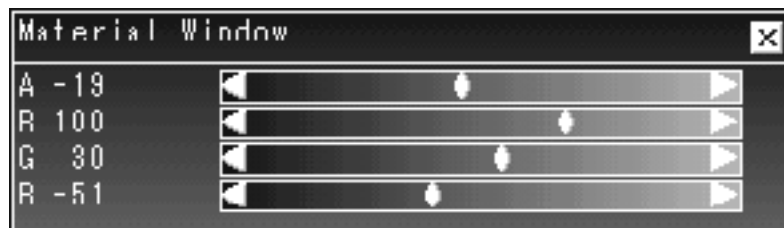


Figure 1.9 An example of creating a scrollbar control

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

Table 2.13 *List of Functions Used in Creating Scroll Bar Controls*

Nindows 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

```
typedef struct {
```

```
    Sint32 n;  
    Sint32 style;  
    Sint16 x, y;  
    Sint16 w, h;  
    NWS_SCROLLBARINFO* info;  
} NWS_SCROLLBARLIST;
```

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.

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 2.14 *List of Low-level Scrollbar Functions*

Nindows API

nwCreateScrollBar

Low-level Scrollbar Function

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

nwSetScrollBarData**Low-level Scrollbar Function**

Format	<code>void nwSetScrollBarData(NHWND hScl, NWS_DATA* data)</code>
Parameters	<code>hScl</code> Window handle of the scrollbar control <code>data</code> 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 hScl, Float min, Float max)</code>
Parameters	<code>hScl</code> Window handle of the scrollbar control <code>min</code> Minimum value of the data associated with the scrollbar <code>max</code> 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	<code>void nwSetScrollBarPos(NHWND hScl, Float pos)</code>
Parameters	<code>hScl</code> Window handle of the scrollbar control <code>pos</code> Value of the data associated with the scrollbar
Return value	None
Function	Sets the value of the data associated with the scrollbar
Reference	<code>nwSetScrollBarRange()</code> , <code>nwSetScrollBarData()</code> , <code>nwSetScrollBarLineMove()</code> , <code>nwSetScrollBarPageMove()</code>
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	<code>void nwSetScrollBarLineMove(NWHWND hScl, Float step)</code>
Parameters	<code>hScl</code> Window handle of the scrollbar control <code>step</code> Step value
Return value	None
Function	Sets the amount of data changed when the scrollbar arrows are pressed
Reference	<code>nwSetScrollBarRange()</code> , <code>nwSetScrollBarData()</code> , <code>nwSetScrollBarPos()</code> , <code>nwSetScrollBarPageMove()</code>
Note	The default is 1.0
Example	<code>// Sets the amount of data changed when the scrollbar arrow is pressed to 2</code> <code>nwSetScrollBarLineMove(2.f);</code>

nwSetScrollBarPageMove

Low-level Scrollbar Function

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

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.

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



Figure 1.10 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.

8.4 Description of Functions for Button Controls

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

Table 2.15 *List of Functions for Creating Button Controls*

Nindows API

nwCreateButton

Button Function

Format `NHWND nwCreateButton(NWF_BUTTONFUNC func, Sint8 *caption, Sint32 x, Sint32 y, Sint32 w, Sint32 h, NHWND 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 `//Creates an "OK" button`
 `NHWND button = nwCreateButton(button_callback_back, "OK", 3, 20, 48, 13, hWndParent);`

nwEnableButton

Button Function

Format `void nwEnableButton(NHWND 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 `// Make a button invalid`
 `nwEnableButton(button, FALSE);`

Callback Function

ButtonCallback

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	

Button Callback Function

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.



Figure 1.11 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.

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

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.



Figure 1.12 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.

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.



Figure 1.13 Condition where the user menu has been entered (2)

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

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.



Figure 1.14 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.

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.

9.5 Description of Functions for Entering User Menus

Function	Description
nwSetUserMenu	Enters a user menu in the System Menu

Table 2.16 *List of Functions for Entering User Menus*

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**

Format	void MenuCallback(NHWND hWnd, Sint32 idx, Sint32 param)						
Parameters	<table><tr><td>hWnd</td><td>Window handle of the menu window where the callback originated</td></tr><tr><td>idx</td><td>Index of the selected menu item in the menu table</td></tr><tr><td>param</td><td>Parameter set in the menu table</td></tr></table>	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
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 selected.	User defined function called back by the menu window when the menu is selected.						
Reference							
Note							
Example							

Menu Function**Structure****NWS_MENUTABLE****Structure**

Definition	<pre>typedef struct _NWS_MENUTABLE { Sint32 type; Sint8 *title; NWF_MENUHANDLE func; Sint32 param; } NWS_MENUTABLE;</pre>									
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	<table><tr><td>type</td><td>Menu item type</td></tr><tr><td>title</td><td>Menu item text</td></tr><tr><td>func</td><td>Callback function for when the menu is selected</td></tr><tr><td>param</td><td>Parameter passed to the callback function</td></tr></table>	type	Menu item type	title	Menu item text	func	Callback function for when the menu is selected	param	Parameter passed to the callback function	
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

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()`.

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



Figure 1.15 *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.

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

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	<pre>NWHWND hWnd = nwCreateMenu(menu_tbl, "MENU", 100, 100, NULL);</pre>

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

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
<code>njGetPeripheral</code>	Gets information about peripherals

Table 2.17 *List of Functions for Entering User Menus*

Nindows API

`njGetPeripheral`

Ninja Function

Format	<code>NJS_PERIPHERAL* njGetPeripheral(long port)</code>
Parameters	<code>port</code> Peripheral port number Please specify <code>NJD_PORT_SYSMOUSE</code> to get information about the mouse
Return value	Address of the structure which stores the mouse information
Function	Gets information about the mouse.
Reference	<code>NJS_PERIPHEAL</code> 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	This structure is not defined by Nindows, it is a Ninja structure. It stores information about joysticks, the keyboard, mouse and other input devices.	
Members	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
Reference	njGetPeripheral()	

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
nwGetFontSize	Get the width and height of a font

Table 2.18 *User Menu Input-Related Function List.*

Nindows API

nwGetFontSize

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	<code>Sint32 width, height; nwGetFontSize(&width, &height);</code>

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.