General Notice

When using this document, keep the following in mind:

1. This document is confidential. By accepting this document you acknowledge that you are bound by the terms set forth in the non-disclosure and confidentiality agreement signed separately and /in the possession of SEGA. If you have not signed such a non-disclosure agreement, please contact SEGA immediately and return this document to SEGA.

2. This document may include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new versions of the document. SEGA may make improvements and/or changes in the product(s) and/or the program(s) described in this document at any time.

3. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without SEGA'S written permission. Request for copies of this document and for technical information about SEGA products must be made to your authorized SEGA Technical Services representative.

4. No license is granted by implication or otherwise under any patents, copyrights, trademarks, or other intellectual property rights of SEGA Enterprises, Ltd., SEGA of America, Inc., or any third party.

5. Software, circuitry, and other examples described herein are meant merely to indicate the characteristics and performance of SEGA's products. SEGA assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples describe herein.

6. It is possible that this document may contain reference to, or information about, SEGA products (development hardware/software) or services that are not provided in countries other than Japan. Such references/information must not be construed to mean that SEGA intends to provide such SEGA products or services in countries other than Japan. Any reference of a SEGA licensed product/program in this document is not intended to state or simply that you can use only SEGA's licensed products/programs. Any functionally equivalent hardware/software can be used instead.

7. SEGA will not be held responsible for any damage to the user that may result from accidents or any other reasons during operation of the user's equipment, or programs according to this document.

NOTE: A reader's comment/correction form is provided with this document. Please address comments to :

SEGA of America, Inc., Developer Technical Support (att. Evelyn Merritt)
150 Shoreline Drive, Redwood City, CA 94065

SEGA may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

(11/2/94- 002)

“SGA3” IFF SEGA Sega 3D Format DRAFT

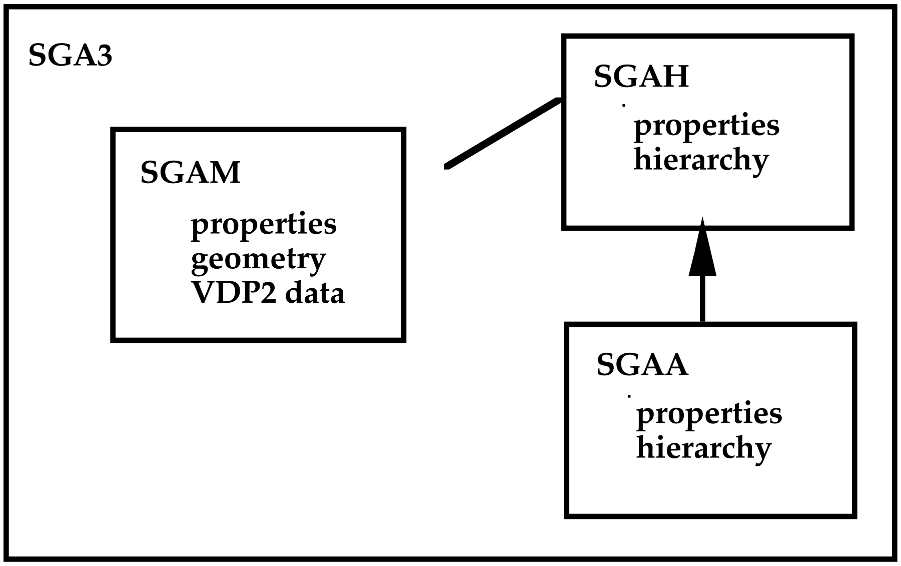
Date: November 9, 1995
From: Jonathan E. Flamm, SEGA of America
Glen Kirk, SEGA of America (Background VDP2 Specification)
dts@segaoa.com
Version: r1.06nv

1.0 Introduction

This document describes the “SGA3” IFF (Interchange Format Files)[1] format for describing three dimensional polygon data and Backgrounds for use on SEGA game platforms. It is the intent of this format to provide a common base for title content for SEGA systems. This will enable the development of tools specifically geared to game development needs. Please refer to reference (1) for a complete description of the IFF format. This document is preliminary and is subject to change.

There are many references in this document to Saturn and Saturn specific hardware features. However, this specification is meant to support other SEGA platforms such as Genesis Super 32X and future hardware. It is the intent of this format to provide a basis for storing both general and platform specific data. As new platforms become available, support will be added to address unique features of the hardware.

The three main forms are SGAM, SGAH, and SGAA. SGA3 is a wrapper form that contains one or more of the preceding forms. SGAM contains the actual geometry data of a model. This includes all the detailed information to depict a polygonal object in three space with all texture and color information. SGAH instantiates SGAM geometries and in addition can describe a hierarchical relationship between model parts. Lastly SGAA provides a means to animate SGAH model hierarchies. Figure 1.0a illustrates this relationship.



---

[1] “EA IFF 85” Standard for Interchange Format Files”, contained in AMIGA ROM Kernel Reference Manual, Devices.

Figure 1.0a. Form categories.

It is important to note that although this format provides the means to create hierarchical animated models, it can be used just as easily for simple flat geometries.

The following is a list of the SGAM, SGAH, and SGAA chunk definitions. A chunk is simply a typed object that contains data -- see section 2.1 for more detail.

SGAM Chunk Categories

MoID	Model ID
MoNm	Model Name
VERT	Vertex List
VerF	Vertex List
NORM	Normal List
NorF	Normal List
TexH	Texture Header
TEXR	Saturn Texture Map Data
TexP	Texture Map Data (Palette)
TexG	Generalized RGB Texture map
GTab	Gouraud table
LINE	Line
PLin	Polyline
Poly	Polygon
TPol	Textured Polygon
TRIG	Triangle
TQTM	Textured quadratic texture map

Pa15	15 bit palette
Pa24	24 bit palette
PalG	Generalized palette
CHR4	4 bit palettized character
CHR8	8 bit palettized character
CH15	15 bit palettized character
RC15	15 bit RGB character
RC24	24 bit RGB character
CMap	Character Map
BMP4	Saturn Bitmaps
BMP8	
BM15	
BM24	

SGAH Chunk Categories

HiID	Hierarchy ID
HiNm	Hierarchy Name
MoID	Model ID
POSN	Position
PosF	Position (floating point)
SCAL	Scale Factor
ScaF	Scale Factor (floating point)
ROTN	Rotation
RotF	Rotation (floating point)

SGAA Chunk Categories

AnNm	Animation Name
AnID	Animation ID
FrRt	Frame Rate
FRAM	Frame Data

1.1 IFF (Interchange Format Files) Introduction

IFF[1] provides a method for storing data in a machine independent fashion. It is a set of conventions for typing data objects. The principal element of IFF is the chunk. A chunk is a typed data object as shown in figure 1.1a.

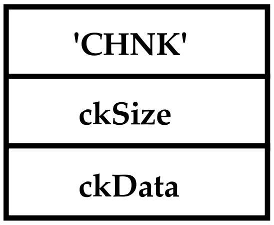


Figure 1.1a. IFF chunk.

Every chunk's type is distinguished by a 4 character header (32 bits). Following the header is a LONG (defined below) ckSize which is the length of data that follows in the ckData section. There are certain conventions that IFF adheres to.

Numbers are big endian. The following are IFF types are defined:

UBYTE	8 bits unsigned
WORD	16 bits signed
UWORD	16 bits unsigned
LONG	32 bits signed

SGA3 adds the following type definitions:

ULONG	32 bits unsigned

CkSize is the size of the data not including the header. All data objects larger than a byte are aligned on even byte boundaries. This may require inserting pad bytes which are to be written as zeroes. Pad bytes after a ckData segment to align the next chunk are not to be counted in the header.

Certain chunk types are optional whereas others are required. Chunk types can be added to a specification and tools can process updated files by interpreting only chunks that they understand. This allows extensibility while maintaining some level of compatibility.

There are several reserved chunk names: FORM, LIST, PROP, CAT, LIS1-LIS9, FOR1-FOR9, CAT1-CAT9.

FORM is a special chunk that contains all the chunks for a defined format. SGAM, SGAH, and SGAA are all forms.

Please see reference 1 for a more rigorous description of IFF, especially if you are writing tools to output or interpret this information.

1.2 SGA3 Guidelines

This section will lay down some rules as to how SGA3 should be used:

- Each SGAM form must be self contained. I.e. if it contains textured polygons, the texture chunks must be present.
- Each SGAM form should contain one model geometry. Multiple geometries should be accomplished by using multiple SGAM chunks. This means that there is only one VERT, VerF, NORM, and NorF chunk per SGAM.
- SGAM forms must have the following chunks in the specified order: MoID, VERT, NORM.
- SGAM MoID numbers must be unique within a SGA3 form.

1.3 Saturn Specific Chunks

The following chunks contain Saturn specific information: GTab, TEXR, Pa15, Pa24, CHR4, CHR8, CH15, RC15, RC24, CMap, BMP4, BMP8, BM15, BM24. These may, however, be used in a general context where appropriate.

1.4 NV Specific Chunks

The following chunks contain NV specific information: TQTM. These may, however, be used in a general context where appropriate.

2.0 SGA3 Conventions

2.1 Coordinate System

By convention we will establish a right handed coordinate system for SGA3. Figure 2.1a details the convention we will follow. The origin is at the top left corner of the screen. Increasing X coordinates follow towards the right of the screen and increasing Y coordinates are towards the bottom of the screen. By taking the cross product of X and Y vectors we see that positive Z is "into the screen."

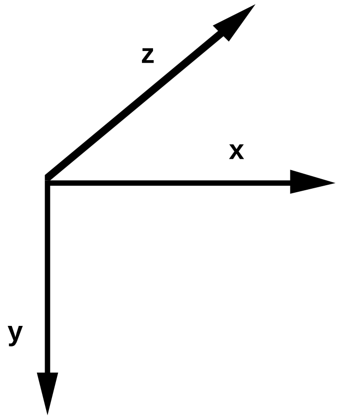


Figure 2.1a. Saturn right handed coordinate system. Positive Z direction is "into" paper. Origin is at upper left corner.

2.2 Part Types

For Saturn we will refer to polygons and lines as parts. Saturn is capable of drawing two point lines and four point polygons. Lines are single color and can optionally have gouraud shading applied to them. The polygonal parts are broken down into filled quadrangles (called polygons), and wire frame quadrangles (called polylines). Filled polygons can contain a solid color or a texture. Each of the polygonal parts can have gouraud shading applied to it. Figure 2.2a illustrates these parts.

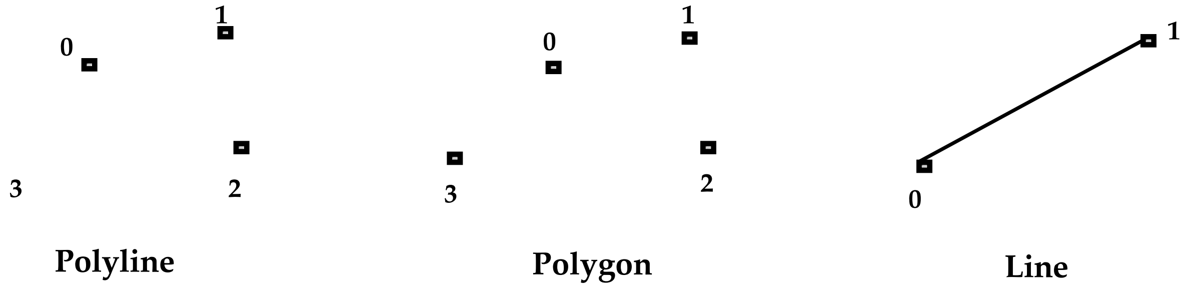


Figure 2.2a. Saturn "parts."

2.3 Vertex Labeling

By convention we will label all vertices in a clockwise direction starting at the point closest to the origin (top left of screen). Figure 2.3a shows the labeling for a line.

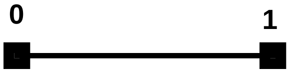


Figure 2.3a. Line, vertices labeled clockwise from top left.

Figure 2.3b shows labeling for any of the polyhedral parts.

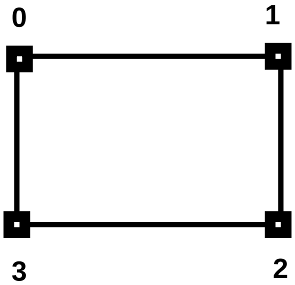


Figure 2.3b. Quadrangle, vertices labeled clockwise from top left.

Figure 2.3c shows labeling for the NV nine point quadratic patch (QTM).

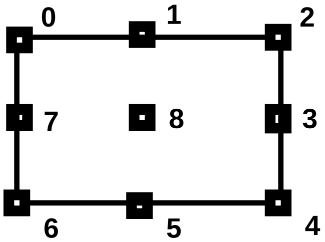


Figure 2.3c. NV quadratic patch, vertices clockwise from top left, center control point is last.

Since Saturn format deals only with quadrangles, triangles must be represented as a degenerate form of a quadrangle. If the triangle is textured, it will matter which vertices are collapsed. Figure 2.3d illustrates an example with the first two vertices collapsed. This would cause a rectangular picture to be squeezed at the top and undistorted at the bottom.

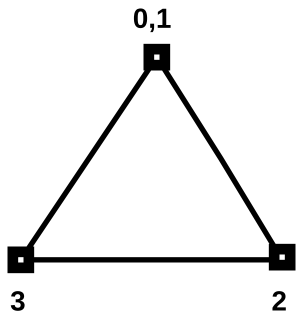


Figure 2.3d. Saturn triangle (degenerate quadrangle), vertices labeled clockwise from top left.

In a three dimensional model, the vertices should always be labeled in a clockwise order if viewed from the outside of the model. This is necessary to create surface normal vectors.

2.4 Number Formats

Sections 2.5 and 2.6 define fixed and floating point number conventions respectively. Many chunks either require fixed or floating point numbers. The convention for SGAM is that the fixed point chunks are required, and the floating point chunks are optional. It is important to note that it is highly recommended to output the floating point chunks even if they are not used in a particular application. The extra precision present in the floating point representation might prove useful for implementation changes or future platform conversions. In general it is better to have the information and not use it then not have it.

2.5 Fixed Point Notation

We introduce a data type *Fixed32* which all vertex and normal coordinates will be defined by:

```
typedef LONG Fixed32;
```

Fixed32 is a 32 bit fixed point format with the low word describing the fractional portion of the number and the high word the whole portion. The binary point is between bit 15 and 16. Figure 2.5a illustrates *Fixed32* .

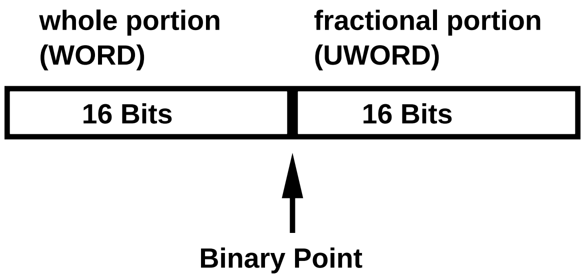


Figure 2.5a. Fixed32 fixed point format.

2.6 Floating Point Notation

We introduce a data type *FLOAT* which some vertex and normal coordinates will be defined by:

```
typedef float FLOAT;
```

FLOAT is a single precision floating point format conforming to the IEEE 754 specification. Figure 2.6a illustrates *FLOAT*.

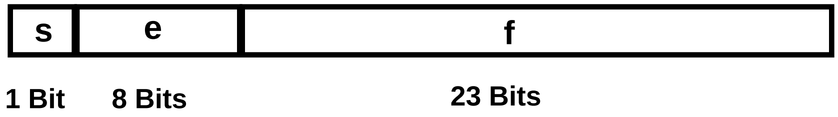


Figure 2.6a. FLOAT floating point format.

In all of the SGAM chunks, floating point numbers are optional but suggested.

2.7 Color Data

Saturn supports both 15 bit and 24 bit RGB color data. RGB data is defined to be 15 bits with the high bit set to 1. The data is not stored as straight RGB but as the following figure 2.7a.

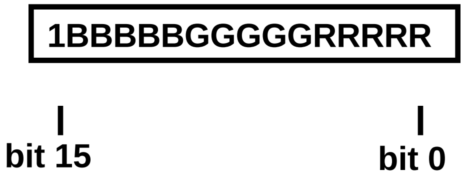


Figure 2.7a. Saturn 15 bit Color Data.

Figure 2.7b indicates 24 bit color format.

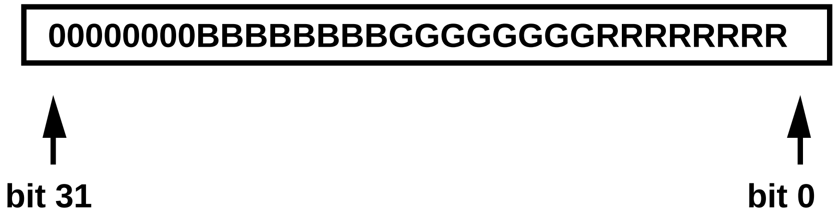


Figure 2.7b. Saturn 24 bit Color Data.

Figure 2.7c illustrates the generalized RGB format that is used in the TexG and PalG chunks. Red, green, and blue components use the same number of bits and unused bits are set to zero. The color data is fit into the minimum number of bytes. Fifteen bit data is fit into two bytes as in figure 2.7c. Twenty four bit data is put into three bytes.

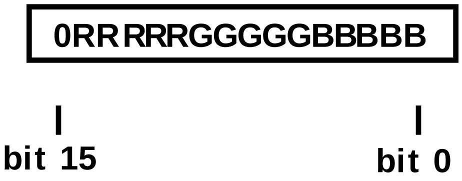


Figure 2.7c. Generalized RGB data 15 bits.

3.0 Model Description Chunk Definitions

3.1 'MoNm'

The 'MoNm' chunk contains a string that is a text name for the model geometry that follows. It is a null terminated string from 1 to 255 characters.

```
char *Name;       /* Null terminated string <= 255 characters */
```

3.2 'MoID'

The 'MoID' property chunk contains the ID number for the model geometry that follows.

```
WORD  ModelID;    /* Model ID number of following model data */
```

3.3 'VERT'

The 'VERT' chunk contains a list of n (x,y,z) triplets representing vertex coordinates. The points are stored as fixed point values as described above. These vertices are mapped to polygons in following data chunks. They are referenced by their index in the vertex list. The VERT contains an array of the Vertex structure defined below. Each vertex has an implicit index from the first element in the list which is defined to be 0.

```
struct Vertex {
      Fixed32     vx;   /* x coordinate of vertex (fixed point) */
      Fixed32     vy;   /* y coordinate of vertex (fixed point) */
      Fixed32     vz;   /* z coordinate of vertex (fixed point) */
};
```

typedef struct Vertex Vertex;
typedef Vertex VertexList[];

3.4 'VerF'

The 'VerF' chunk is similar to 'VERT' except floating point values are used. This is an optional but recommended chunk to include in the 'SGAM' specification.

```
struct VertexFp {
      FLOAT vx;         /* x coordinate of vertex (floating point) */
      FLOAT vy;         /* y coordinate of vertex (floating point) */
      FLOAT vz;         /* z coordinate of vertex (floating point) */
};
```

typedef struct VertexFp VertexFp;
typedef VertexFp VertexListFp[];

3.5 'NORM'

The 'NORM' chunk contains a list of n (x,y,z) triplets representing vertex normal vectors. The normals must have a one to one correspondence with the vertices as the correlation between vertex and normal is determined by matching indices. The points are stored as fixed point values as described above. The normals are used in polygons below. They are referenced by their index in the normal list.

```
struct Normal_Data {
      Fixed32     x;
      Fixed32     y;
      Fixed32     z;
};

typedef struct Normal Normal ;
typedef Normal Normal List[];
```

3.6 'NorF'

The ‘NorF’ chunk is similar to ‘NORM’ except floating point values are used. This is an optional but recommended chunk to include in the ‘SGAM’ specification.

```
struct Normal_DataFp {
      FLOAT x;
      FLOAT y;
      FLOAT z;
};

typedef struct NormalFp NormalFp ;
typedef NormalFp Normal ListFp[];
```

3.7 ‘TexH’

The ‘TexH’ chunk contains a string that is a text name for the texture that follows. It is a null terminated string from 1 to 255 characters.

```
char *Name;       /* Null terminated string <= 255
                        characters */
```

3.8 ‘TEXR’

The 'TEXR' chunk contains an RGB texture definition and the texture data defined in 15 bit Saturn format. There is one texture per 'TEXR' chunk. These will be referenced by their ID number. The level of detail field allows different texture sizes to map to a single texture ID. A texture with only one level of detail should have level set to zero. For decreasing levels of detail the level number should increase. This is useful to allow larger or smaller texture sizes to be used depending on the size of the object in the viewing field. Optimal viewing occurs when the texture size is close to the part size to which it is mapped.

```
struct TextureRGB {
      WORD  id;          /* Id # of this texture */
      WORD  level;       /* Level of detail for this texture */
      WORD  width;       /* Width in pixels (must be multiple of 8 for
                            Saturn, power of 2 for NV) */
      WORD  height;      /* Height in pixels (power of 2 for NV) */
      UWORD data[];      /* Binary RGB data stored in Saturn format */
};

typedef struct Texture Texture ;
```

3.9 ‘TexP’

The 'TexP' chunk is similar to the ‘TEXR’ chunk above except it contains palettized color data. The data size is determined by the width and height fields. The paletteID references a palette chunk in section 5.

```
struct TextureP {
      WORD  id;         /* Id # of this texture */
      WORD  level;      /* Level of detail for this texture */
      WORD  width;      /* Width in pixels (must be multiple of 8 for
                           Saturn, power of 2 for NV) */
      WORD  height;     /* Height in pixels (power of 2 for NV)*/
      WORD  paletteID   /* ID of referenced palette */
      UWORD data[];           /* Texture data consists of palette
indices */
};

typedef struct TextureP TextureP ;
```

3.10 ‘TexG’

The 'TexG' chunk contains an RGB texture definition and the texture data defined in general RGB color format. The pixel depth of the data is specified and the data is stored in the minimum number of whole bytes. Fifteen bit data will be stored in two bytes per color. Twenty four bit data will be stored in three bytes per color. TexG follows the same rules as TEXR.

```
struct TextureG {
      WORD  id;         /* Id # of this texture */
      WORD  level;      /* Level of detail for this texture */
      WORD  width;      /* Width in pixels (must be multiple of 8 for
                           Saturn, power of 2 for NV) */
      WORD  height;     /* Height in pixels (power of 2 for NV) */
      WORD  pixelDepth; /* Depth of pixels in bits */
      UBYTE data[];     /* Binary data stored in general RGB format */
};

typedef struct TextureG TextureG ;
```

3.11 ‘GTab’

GTab represents a Saturn format gouraud table. Gouraud tables can be associated with any part. They consist of four offset values for the corners of a quadrangle. See the Saturn VDP1 manual for details. The id variable is a reference to the associated part.

```
struct GouraudTable {
      WORD id;
      WORD table[4];
};
```

typedef struct GouraudTable GouraudTable;

4.0 Part Chunk Definitions

The following chunks can be placed in any order. Each chunk can contain one or more parts.

The Light variable present in all of the part chunks indicates if the part is affected by the lighting model or not. A value of zero will indicate no light source whereas one will indicate lighting. Future versions of the specification may include enumerated types to specify different lighting types. For this reason, this field should not be used except as specified.

Mode is machine specific data, used to represent the Saturn mode bits for VDP1.

Color is specified by referencing an index in a palette (PA15, PA24, or future defined palette chunk). The paletteID variable specifies one of the PA15 or PA24 chunks, and the color variable is an index into this palette. This allows generality in color format. If other platforms use different color formats, the palette chunks can be modified while retaining the original part chunks. Also new palette chunks can be added to support different color schemes.

On Saturn, RGB parts don’t reference palettes. One suggested way of storing Saturn parts is to create a global palette of type PA15 that contains all the colors used by monocolor parts and reference this. In a run-time format, the colors would be collapsed into the parts themselves.

4.1 ‘LINE’

The 'LINE' chunk describes one or more lines described by two points and a color. All vertices are indices into the vertex list in the 'VERT' chunk.

```
struct Line {
      WORD  ID;         /* ID number of this part, Ids start at 1 */
      WORD  Vertex[2];  /* a and b vertices, (indices to vertex list)
*/
      WORD  color;      /* Palette index */
      ULONG mode;       /* reserved data */
      WORD  paletteID   /* ID of palette */
      WORD  light;      /* Use light model */
};

typedef struct Line Line ;
typedef Line Lines[];
```

4.2 ‘PLin’

The 'PLin' chunk describes one or more polyline. A polyline is a quadrangle that is not filled. Four vertices and a color are necessary to describe a polyline on Saturn.

```
struct PolyLine {
      WORD  ID;          /* ID number of this part, Ids start at 1 */
      WORD  Vertex[4];  /* vertices 0-3, (index to vertex list) */
      WORD  color;       /* Palette index */
      ULONG mode;        /* reserved data */
      WORD  paletteID    /* ID of palette */
      WORD  light;       /* Use light model */
};
```

typedef struct PolyLine PolyLine ;
typedef PolyLine PolyLines[];

4.3 'Poly'

Chunk 'Poly' is a solid filled 'PLin'.

```
struct Polygon {
      WORD  ID;          /* ID number of this part, Ids start at 1 */
      WORD  Vertex[4];  /* vertices 0-3, (index to vertex list) */
      WORD  color;       /* Palette index */
      ULONG mode;        /* reserved data */
      WORD  paletteID    /* ID of palette */
      WORD  light;       /* Use light model */
};

typedef struct Polygon Polygon ;
typedef Polygon Polygons[];
```

4.4 'TPol'

Chunk 'TPol' represents a textured polygon. It differs from a normal polygon ('Poly') in that the color is replaced by an index to a texture.

```
struct TexturedPoly {
      WORD  ID;          /* ID number of this part, Ids start at 1 */
      WORD  Vertex[4];  /* vertices 0-3, (index to vertex list) */
      WORD  texture;     /* texture, (texture ID number)) */
      ULONG mode;        /* reserved data */
      WORD  light;       /* Use light model */
};

typedef struct TexturedPoly TexturedPoly ;
typedef TexturedPoly TexturedPolys[] ;
```

4.5 'TRIG'

Chunk 'TRIG' is a solid filled triangle.

```
struct Triangle {
      WORD  ID;          /* ID number of this part, Ids start at 1 */
      WORD  Vertex[3];  /* vertices 0-2, (index to vertex list) */
      WORD  color;      /* Palette index */
      ULONG mode;       /* reserved data */
      WORD  paletteID   /* ID of palette */
      WORD  light;      /* Use light model */
};

typedef struct Triangle Triangle ;
typedef Triangle Triangles[] ;
```

4.6 'TQTM'

Chunk 'TQTM' represents a quadrangle with five control points.

```
struct TQTM {
      WORD  ID;          /* ID number of this part, Ids start at 1 */
      WORD  Vertex[9];  /* vertices 0-8, (index to vertex list) */
      WORD  texture;    /* texture, (texture ID number)) */
      ULONG mode;       /* reserved data */
      WORD  light;      /* Use light model */
};

typedef struct TQTM TQTM ;
typedef TQTM TQTMs[];
```

5.0 Background Types

5.1 'Pa15'

The 'Pa15' chunk contains a palette of 15 bit color values. The high (16th) bit should always be set to 1

```
struct Palette15 {
      WORD  id;          /* ID of this palette, should be >= 1 */
      LONG  index;       /* starting color index */
      LONG  numColors;  /* Number of colors in this palette */
      UWORD Color15[numColors];     /* BGR Color value Bit 15 = 1 */
};

typedef struct Palette15 Palette15 ;
```

5.2 'Pa24'

The 'Pa24' chunk contains a palette of 24 bit color values

```
struct Palette24 {
      WORD  id;               /* ID of this palette, should be >= 1 */
      LONG  index;      /* starting color index */
      LONG  numColors;        /* Number of colors in this palette */
      ULONG Color24[numColors];     /* BGR Color value */
};

typedef struct Palette24 Palette24 ;
```

5.3 ‘PalG’

The 'PalG' chunk is a general RGB palette chunk. Pixel depth is specified and each color is stored in general RGB format. Unused bits are set to zero and the minimum number of bytes is used to store each color.

```
struct PaletteG {
      WORD  id;               /* ID of this palette, should be >= 1 */
      LONG  index;            /* starting color index */
      LONG  numColors;        /* Number of colors in this palette */
      WORD  pixelDepth;       /* Pixel depth in bits */
      UBYTE Color[];          /* General RGB color values */
};

typedef struct PaletteG PaletteG ;
```

5.4 ‘CP04’

Chunk 'CP04' represents palettized 4 bit per pixel character data.

```
struct PaletteChar4
{
      WORD ID;                /* id # of this character list */
      WORD  coidId;           /* id # of the palette this group uses */
      WORD  startId;          /* char # offset first char in list */
      WORD  numChars;         /* # of 8x8 pixel characters */
      UBYTE data[numChars*32];      /* actual pixel map of chars */
};

typedef struct PaletteChar4 PaletteChar4;
```

5.5 ‘CP08’

Chunk 'CPO8' represents palettized 8 bit per pixel character data.

```
struct PaletteChar8
{
      WORD ID;                /* id # of this character list */
      WORD  coidId;            /* id # of the palette this group uses */
      WORD  startId;           /* char # offset first char in list */
      WORD  numChars;          /* # of 8x8 pixel characters */
      UBYTE data[numChars*64];      /* actual pixel map of chars */
};

typedef struct PaletteChar8 PaletteChar8;
```

5.6 'CP11'

Chunk 'CHR11' represents palettized 11 bit per pixel character data stored in 16 bits.

```
struct PaletteChar11
{
      WORD  ID;                /* id # of this character list */
      WORD  coidId;            /* id # of the palette this group uses */
      WORD  startId;           /* char # offset first char in list */
      WORD  numChars;          /* # of 8x8 pixel characters */
      UBYTE data[numChars*128];     /* actual pixel map of chars */
};

typedef struct PaletteChar11 PaletteChar11;
```

5.7 'RC15'

Chunk 'RC15' represents 15 bit per pixel BGR color data.

```
struct RGBChar15
{
      WORD ID;                 /* id # of this character list */
      WORD  coidId;            /* id # of the palette this group uses */
      WORD  startId;           /* char # offset first char in list */
      WORD  numChars;          /* # of 8x8 pixel characters */
      UBYTE data[numChars*128];     /* actual pixel map of chars */
};

typedef struct RGBChar15 RGBChar15;
```

5.8 'RC24'

Chunk 'RC24' represents 24 bit per pixel BGR color data. Format is described in section 2.6.

```
struct RGBChar24
{
      WORD ID;                 /* id # of this character list */
      WORD  coidId;            /* id # of the palette this group uses */
      WORD  startId;           /* char # offset first char in list */
      WORD  numChars;          /* # of 8x8 pixel characters */
      UBYTE data[numChars*256];     /* actual pixel map of chars */
};

typedef struct RGBChar24 RGBChar24;
```

5.9 ‘CMap’

‘CMap’ is a map of character data in relative numbers (i.e., 0 means character 0, 1 means character 1 etc.)

```
struct CharMap
{
      WORD ID;                 /* id # of this map list */
      WORD CharId;             /* id # of the associated char patterns
*/
      WORD width;              /* width in 8*8 characters */
      WORD height;             /* Height in 8*8 characters */
      WORD data[width*height]; /* map of characters laid out in a
linear fashion */
};

typedef struct CharMap CharMap;
```

map data format is as follows:
bit 15: flip vertical: 0=no flip; 1=flip
bit 14: flip horizontal : 0=no flip; 1=flip
bits 13-0: character number

5.10 ‘BMP4’

‘BMP4’ is a saturn palette bitmap data chunk that contains the data for a Saturn Bitmap.

```
struct SPBITMAP4
{
      WORD ID;          /* id # of this chunk */
      WORD coidId;      /* id # of the palette chunk this bitmap */
      WORD width;       /*width of bitmap; valid:  512 or 1024 */
      WORD height;      /* height of bitmap; valid:   256 or 512 */
      UBYTE data[(width*height)/2]; /* actual data
                                    bit format follows that specified
                                    in vdp2 manual pgs 96-110 */
};

typedef struct SPBITMAP4 SPBITMAP4;
```

5.11 ‘BMP8’

‘BMP8’ is a saturn palette bitmap data chunk that contains the data for a Saturn Bitmap.

```
struct SPBITMAP8
{
      WORD ID;            /* id # of this chunk */
      WORD coidId;        /* id # of the palette chunk this bitmap */
      WORD width;         /*width of bitmap; valid:  512 or 1024 */
      WORD height;        /* height of bitmap; valid:   256 or 512 */
      UBYTE data[(width*height)];   /* actual data
                                       bit format follows that specified
                                       in vdp2 manual pgs 96-110 */
};

typedef struct SPBITMAP8 SPBITMAP8;
```

5.12 ‘BM15’

‘BM15’ is a saturn palette bitmap data chunk that contains the data for a Saturn Bitmap.

```
struct SPBITMAP15
{
      WORD ID;            /* id # of this chunk */
      WORD coidId;        /* id # of the palette chunk this bitmap */
      WORD width;         /*width of bitmap; valid:  512 or 1024 */
      WORD height;        /* height of bitmap; valid:   256 or 512 */
      UBYTE data[(width*height)*2]; /* actual data
                                       bit format follows that specified
                                       in vdp2 manual pg. 96-110 */
};

typedef struct SPBITMAP15 SPBITMAP15;
```

5.13 ‘BM24’

‘BM24’ is a saturn palette bitmap data chunk that contains the data for a Saturn Bitmap.

```
struct SPBITMAP24
{
      WORD ID;            /* id # of this chunk */
      WORD coidId;        /* id # of the palette chunk this bitmap */
      WORD width;         /*width of bitmap; valid:  512 or 1024 */
      WORD height;        /* height of bitmap; valid:   256 or 512 */
      UBYTE data[(width*height)*4]; /* actual data size in bytes
                                       bit format follows that specified
                                       in vdp2 manual pg. 96-110 */
};
```

```
typedef struct SPBITMAP24 SPBITMAP24;
```

6.0 Hierarchy

'SGAH' supports a hierarchy of objects. ‘SGAH’ contains references to ‘SGAM’ model objects. For every referenced ‘SGAM’ form (referenced by ID) a ‘POSN’, ‘SCAL’, and ‘ROTN’ chunk must follow. The ‘SGAH’ form can contain embedded ‘SGAH’ forms for each level of hierarchy. Figure 6.0a illustrates this scheme.

```
FORM SGA3

    FORM SGAM
        MoNm
        MoID
        Vert
        Norm
        ...
    FORM SGAM
        MoNm
        MoID
        Vert
        Norm
        ...
    FORM SGAH
        MoID
        POSN
        SCAL
        Rotn

        MoID
        POSN
        SCAL
        ROTN

        FORM SGAH
            MoID
            POSN
            SCAL
            ROTN
```

Figure 6.0a. Hierarchy example.

6.1 ‘HiNm’

The ‘HiNm’ chunk contains a string that is a text name for the hierarchy that follows. It is a null terminated string from 1 to 255 characters.

```
char *Name;       /* Null terminated string <= 255
                  characters */
```

6.2 ‘HiID’

The 'HiID' property chunk contains the ID number for the hierarchy that follows.

```
WORD  HierarchyID;      /* Hierarchy ID number of referenced
                        model data */
```

6.3 ‘MoID’

The 'MoID' property chunk contains the ID number for the model geometry that this hierarchy is referencing.

```
WORD  ModelID;    /* Model ID number of referenced model data */
```

6.4 'POSN'

The 'POSN' property chunk contains an optional position coordinate for the 3D model being described. This indicates an initial translation for this object. The 3D model should be described such that the center is at the origin and this will indicate its initial offset.

```
struct Position = {
      Fixed32     x;    /* x coordinate initial position
                           (fixed point) */
      Fixed32     y;    /* y coordinate initial position
                           (fixed point) */
      Fixed32     z;    /* z coordinate initial position
                           (fixed point) */
};

typedef struct Position Position ;
```

6.5 'PosF'

The ‘PosF’ chunk is similar to ‘POSN’ except floating point values are used. This is an optional but recommended chunk to include in the ‘SGAM’ specification.

```
struct PositionFp = {
      FLOAT x;          /* x coordinate initial position
                           (floating point) */
      FLOAT y;          /* y coordinate initial position
                           (floating point) */
      FLOAT z;          /* z coordinate initial position
                           (floating point) */
};

typedef struct PositionFp PositionFp ;
```

6.6 'SCAL'

The 'SCAL' property chunk contains optional scaling values for the 3D model being described. This indicates an initial scaling factor for this object. Each of the three cartesian coordinates in the NORM chunks will be multiplied by the corresponding scaling factor. The default scaling factor is 1.

```
struct Scale = {
      Fixed32     x;    /* x scale factor (fixed point) */
      Fixed32     y;    /* y scale factor (fixed point) */
      Fixed32     z;    /* z scale factor (fixed point) */
};

typedef struct Scale Scale ;
```

6.7 'ScaF'

The ‘ScaF’ chunk is similar to ‘SCAL’ except floating point values are used. This is an optional but recommended chunk to include in the ‘SGAM’ specification.

```
struct ScaleFp = {
      FLOAT x;          /* x scale factor (floating point) */
      FLOAT y;          /* y scale factor (floating point) */
      FLOAT z;          /* z scale factor (floating point) */
};

typedef struct ScaleFp ScaleFp ;
```

6.8 'ROTN'

The 'ROTN' property chunk contains an optional rotation values for the 3D model being described. This indicates an initial rotation value for this object. The default rotation is 0.

```
struct Rotation = {
      Fixed32     x;    /* x rotation factor (Fixed point) */
      Fixed32     y;    /* y rotation factor (Fixed point) */
      Fixed32     z;    /* z rotation factor (Fixed point) */
};

typedef struct Rotation Rotation ;
```

6.9 'RotF'

The 'RotF' chunk is similar to 'ROTN' except floating point values are used. This is an optional but recommended chunk to include in the 'SGAM' specification.

```
struct RotationFp = {
      FLOAT x;          /* x rotation factor (floating point) */
      FLOAT y;          /* y rotation factor (floating point) */
      FLOAT z;          /* z rotation factor (floating point) */
};

typedef struct RotationFp RotationFp ;
```

7.0 Animation

Animation is represented by the FORM 'SGAA'. The format is designed to be simple. Each animation FORM contains a name and ID to identify the animation sequence followed by the frame rate, and then the actual frames.

The frames consist of a frame number followed by a list of hierarchies and their position, scale, and rotation vectors. Only those hierarchies that have changed since the last animation frame need to be referenced. Only the pieces of the hierarchy that are affected need to be referenced. Figure 7.0a shows an example of the 'SGAA' FORM.

```
FORM SGAA

      AnNm
      AnID
      FRRT

      FRAM

      FRAM
      ...
```

Figure 7.0a. Animation example.

7.1 'AnNm'

The ‘AnNm’ chunk contains a string that is a text name for this animation sequence. It is a null terminated string from 1 to 255 characters.

```
char *Name;       /* Null terminated string <= 255
                     characters */
```

7.2 ‘AnID’

The 'AnID' property chunk contains the ID number for this animation sequence.

```
WORD  Animation ID;     /*ID number of animation sequence */
```

7.3 ‘FRRT’

The 'FRRT' property chunk contains the frame rate of this animation. It is an integer value indicating 1/60s of a second.

```
WORD  FrameRate;  /* Frame rate in 1/60s of a second */
```

7.4 ‘FRAM’

The chunk ‘FRAM’ is a list of hierarchy references and their associated position, scale, and rotation vectors. These hierarchies represent those elements that have changed since the last animation frame.

```
struct FrameElement = {
      WORD  HierarchyID;      /* Hierarchy ID number of referenced
                           model data */
      Positon     pos;
      Scale       scale;
      Rotation    rotaton;
};

typedef struct FrameElement FrameElement;

struct Frame = {
      WORD              FrameNum;         /* Number of this
                                             animation  frame */
      FrameElement      elements[N];      /* List of frame elements */
};

typedef struct Frame Frame;
```

8.0 SGAM Regular Expression

```
SGAM            ::= "FORM" #{     "SGAM"  [MoNm] MoID VERT [VerF]
                                  NORM [NorF] [TexH] [TEXR] [TexP]
                                  [TexG] [GTab] [LINE] [Plin] [Poly]
                                  [Tpol] [TRIG] [TQTM] [Pa15] [Pa24]
                                  [PalG] [CHR4] [CHR8] [CH15] [RC15]
                                  [RC24] [CMap] [BMP4] [BMP8] [BM15]
                              }

MoNm      ::= "MoNm"  #{ Name }
MoID      ::= "MoID"  #{ ModelID }
VERT      ::= "VERT"  #{ Vertex* }
VerF      ::= "VerF"  #{ VertexFp* }
NORM      ::= "NORM"  #{ Normal* }
NorF      ::= "NorF"  #{ NormalFp* }
TexH      ::= "TexH"  #{ Name }
TEXR      ::= "TEXR"  #{ Texture }
TexP      ::= "TexP"  #{ TextureP }
TexG      ::= "TexG"  #{ TextureG }
GTab      ::= "GTab"  #{ GouraudTable* }
LINE      ::= "LINE"  #{ Line* }
PLin      ::= "PLin"  #{ PolyLine* }
Poly      ::= "Poly"  #{ Polygon* }
TPol      ::= "TPol"  #{ TexturedPoly* }
TRIG      ::= "TRIG"  #{ Triangle* }
TQTM      ::= "TQTM"  #{ TQTM* }
Pa15      ::= "Pa15"  #{ Palette15 }
Pa24      ::= "Pa24"  #{ Palette24 }
PalG      ::= "PalG"  #{ PaletteG }
CHR4      ::= "CHR4"  #{ PaletteChar4 }
CHR8      ::= "CHR8"  #{ PaletteChar8 }
CH15      ::= "CH15"  #{ PaletteChar15 }
RC15      ::= "RC15"  #{ RGBChar15 }
RC24      ::= "RC24"  #{ RGBChar24 }
CMap      ::= "CMap"  #{ CharMap}
BMP8      ::= "BMP8"  #{ SPBITMAP8 }
BM15      ::= "BM15"  #{ SPBITMAP15 }
BM24      ::= "BM25"  #{ SPBITMAP24 }
```

Literal strings are shown in "quotes." The token # represents a ckSize LONG count of the following braced items. The token "*" denotes 0 or more repetitions of an item.

example:

To describe two vertices, the following data would be used:

'"VERT"
24
Vertex[2]

First the literal "VERT," then ckSize = 24 which is 2 * sizeof(Vertex), and then the actual vertex data.

The chunks: LINE, Plin, Poly, PoCG, TPol, TRIG, and TQTM can be in any order but must follow those defined above.

9.0 SGAH Regular Expression

```
SGAH ::= “FORM” #{        “SGAH”  [HiNm] HiID MoID POSN [PosF] SCAL
                          [ScaF] ROTN [RotF] [SGAH]
                          }
HiNm ::= “HiNm”   #{ Name }
HiID         ::= “HiID”     #{ HierarchyID }
MoID         ::= “MoID”     #{ ModelID }
POSN ::= “POSN”   #{ Position }
SCAL         ::= “SCAL”     #{ Scale }
ROTN ::= “ROTN”   #{ Rotation }
```

10.0 SGAA Regular Expression

```
SGAA ::= “FORM” #{        “SGAA”  [AnNm] AnID FRRT FRAM     }

AnNm         ::= “AnNm”  #{ Name }
AnID         ::= “AnID”     #{ AnimationID }
FRRT         ::= “FRRT”     #{ FrameRate }
FRAM ::= “FRAM”   #{ Frame }
```