



Dreamcast™

**ハードウェア概要**

**Version 1.00**

**Last Update : 98/10/15**

**(株)セガ・エンタープライゼス**

## 目次

<b>§1 ハードウェアの構成</b>	<b>4</b>
§1.1 CPU	4
§1.2 グラフィック	4
§1.3 サウンド	4
§1.4 メモリ	4
§1.5 GD-ROM ドライブ	4
§1.6 MODEM	4
<b>§2 各モジュールの概要</b>	<b>5</b>
<b>§3 グラフィック機能</b>	<b>6</b>
§3.1 HOLLY	6
§3.2 HOLLY システム構成	6
§3.3 POWERVR2 とは？	6
§3.4 POWERVR2 ブロック構成	7
§3.5 タイル分割処理の概念	7
§3.6 タイルアクセラレータによる描画のしくみ	8
§3.7 ポリゴンリスト	9
§3.8 ストリップ	9
§3.9 SORTMODE	10
§3.10 頂点カラー	11
§3.11 MODIFIER VOLUME ( モディファイアボリューム )	11
§3.12 MIPMAP	12
§3.13 アルファブレンディング ( 混合処理 )	13
§3.14 バンプマッピング	14
§3.15 テクスチャフォーマット	15
§3.15.1 ピクセルデータ概略	15
§3.15.2 RGB テクスチャ	15
§3.15.3 YUV テクスチャ	15
§3.15.4 Bump Map テクスチャ	16
§3.15.5 Palette テクスチャ	16
§3.15.6 テクスチャデータサイズ	17
§3.15.7 テクスチャデータ形式	17
§3.15.8 VQ テクスチャ圧縮	19
§3.16 テクスチャフィルタ	20
§3.16.1 Point Sampling ( ポイントサンプリング )	20
§3.16.2 Bi-Linear Filter ( バイリニアフィルタ )	21
§3.16.3 Tri-Linear Filter ( トライリニアフィルタ )	22
§3.16.4 Texture Super Sampling ( テクスチャスーパーサンプリング )	22
§3.16.5 各種テクスチャフィルタの画面効果	23
§3.17 フォグ	24
§3.18 クリッピング	24
§3.18.1 タイルクリッピング	24
§3.18.2 ピクセルクリッピング	26
§3.19 バッファ	27

---

§3.19.1 Native Buffer (ネイティブバッファ) .....	27
§3.19.2 Native Buffer の容量.....	27
§3.19.3 Texture Buffer (テクスチャバッファ) .....	27
§3.19.4 Accumulation Buffer (アキュムレーションバッファ) .....	28
§3.19.5 Frame Buffer (フレームバッファ) .....	28
§3.19.6 Frame Buffer サイズの算出方法.....	28
§3.19.7 Strip Buffer (ストリップバッファ) .....	28
§3.20 IMAGE SUPER SAMPLING .....	29
§3.21 フリッカフリーフィルタリング .....	30
§3.21.1 タイプ A .....	30
§3.21.2 タイプ B .....	31
§3.22 カラー拡張とディザ処理.....	31
§3.23 カラークランプ .....	31
§3.24 RENDER TEXTURE (レンダーテクスチャ) .....	32
§3.25 デジタルビデオエンコーダ .....	32
<b>§4 サウンド機能.....</b>	<b>33</b>
§4.1 サウンド機能の概要 .....	33
§4.2 ADPCM とは? .....	33
<b>§5 ペリフェラル.....</b>	<b>36</b>
§5.1 ビジュアルメモリ.....	36
§5.2 標準コントローラー .....	36
§5.3 対応予定のペロフェラル.....	37
<b>§6 GD-ROM (仮称).....</b>	<b>39</b>
§6.1 GD-ROM とは? .....	39
§6.2 GD-ROM の構造.....	39
<b>§7 CPU .....</b>	<b>40</b>
§7.1 FPU.....	40
§7.2 MMU.....	40
§7.3 DMA.....	41
§7.4 タイマ .....	41
§7.5 キャッシュ.....	42
§7.6 ストアキュー .....	42
§7.7 パイプライン・スーパースカラ .....	42
<b>§8 MODEM .....</b>	<b>46</b>

---

## §1 ハードウェアの構成

Dreamcast のハードウェアは大きく分けて次のようなモジュールに分けることができます。

- ・CPU (Hitachi SH7091 カスタム 200MHz)
- ・グラフィック (Holly)
- ・サウンド (AICA)
- ・メモリ (システムメモリ 16MB、テクスチャ RAM 8 MB、サウンド RAM 2MB)
- ・GD-ROM ドライブ (GigaByteDisk ROM ドライブ: 仮称)
- ・Modem

それぞれについて簡単に説明します。

### §1.1 CPU

CPU として、200MHz で動作する日立製作所製の SH4 をベースに、セガ仕様にカスタマイズした SH7091 マイコンを使用しています。

### §1.2 グラフィック

グラフィック LSI として Holly を搭載していますが、この LSI はグラフィックコントロール部分 (PowerVR) とシステムコントローラを含んだ構造になっています。SATURN の VDP1 と SCU、SMPC の機能がひとつになったものと見ることもできます。

### §1.3 サウンド

サウンド用の LSI には、YAMAHA 製の AICA を搭載しています。

AICA には、サウンド処理を行うための部分以外にサウンドコントロール用として、Advanced RISC Machines (ARM) 社製の ARM7DI を内蔵しており、この LSI だけで、サウンドに関するすべての仕事ができます。

### §1.4 メモリ

メモリは、メイン RAM として 16MB、テクスチャデータや、フレームバッファ、ディスプレイリスト用に 8MB、サウンド用に 2MB 積んでいます。

いずれも、シンクロナス DRAM という高速なアクセスの可能な RAM を使っています。

### §1.5 GD-ROM ドライブ

ソフトの供給元として 1 Giga Byte CD-ROM という Dreamcast 専用メディアを読むことができる GD-ROM ドライブを搭載しています。

GD-ROM は、ディスクの回転速度が一定の方式であるため、ディスクの内周部と外周部では読み込み速度が異なり、4 倍速から 12 倍速の読み込み速度になります。

### §1.6 Modem

33.6Kbps のソフトウェアモデムを搭載しています。

## **§2 各モジュールの概要**

次のモジュールについてより詳しく説明します。

- ・グラフィック機能
- ・サウンド機能
- ・ペリフェラル
- ・GD-ROM
- ・CPU
- ・Modem

## § 3 グラフィック機能

Dreamcast のグラフィック機能は Holly チップ内の PowerVR 2 という部分で取り扱われます。  
PowerVR2 は、複雑な 3D モデルを効率よく扱うことができるように設計されており、さまざまな機能を持っています。

### § 3.1 Holly

Holly は、(株)SEGA と(株)NEC、(株)ビデオロジックで共同開発されたカスタム IC です。  
システム構成は、SH4 CPU とのインターフェースバスコントローラ、タイルアクセラレータ、PowerVR2 コアからなります。  
本項では、PowerVR2 に関連するブロックについて解説します。

### § 3.2 Holly システム構成

Holly 内部のシステム構成を以下に示します。

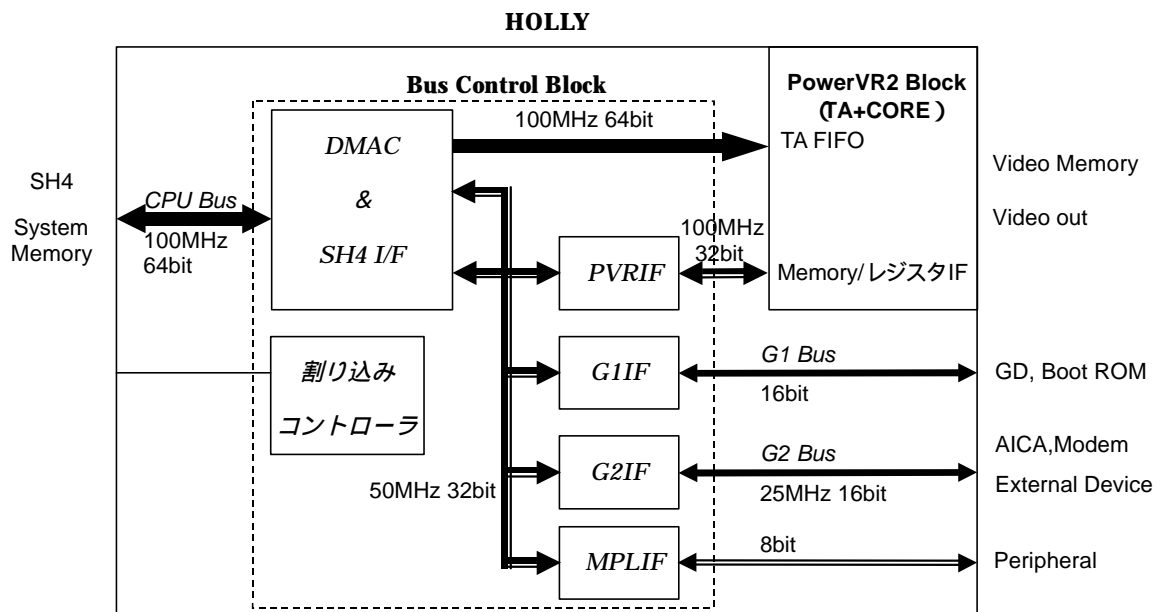


図 3-1Holly 内部ブロック概要

### § 3.3 PowerVR2 とは？

PowerVR2 は NEC 社とイギリスのビデオロジック社の共同開発により完成した新世代の 3D 描画アーキテクチャです。主な特徴として

- ( 1 ) 従来処理フローと異なり、ポリゴンを上書きしてしまうという無駄な処理をなくします。
- ( 2 ) バッファ・メモリを使用しない独自開発の隠面消去技術と処理効率を追求した処理フローにより、従来必要とされていた高価なメモリの使用とバンド幅のボトルネックから解放され、非常にコストパフォーマンスの高いシステムが実現できます。
- ( 3 ) 無限平面モデリング採用の 3D レンダリング・アルゴリズムは、従来(ポリゴン・メッシュ)に較べてデータ量、処理量の大幅に軽減でき、また、リアルな影の生成を容易にし、3D グラフィクスをよりリアルに表現します。

まさに、家庭用ゲーム機向けのアーキテクチャといえます。これら特徴は PowerVR2 独自のアーキテクチャであるところのタイル分割処理により実現されます。

### § 3.4 PowerVR2 ブロック構成

PowerVR2 ブロックの内部構成(概略)を以下に示します。

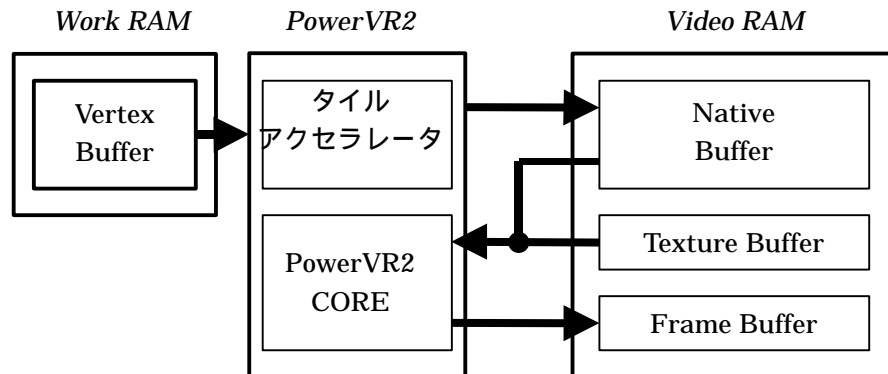


図 3-2 PowerVR2 ブロック構成

- Vertex Buffer** :ポリゴンリスト情報を格納します。(CPU によって Work RAM 上に設定されます)
- タイルアクセラレータ** :ポリゴンリストをタイル分割して Native Buffer に格納します。
- Native Buffer** :PowerVR2 CORE で処理されるディスプレイリスト情報を格納します。
- PowerVR2 CORE** :Sort, 描画処理を行い、結果を Frame Buffer にコピーします。
- Texture Buffer** :テクスチャデータを格納します。
- Frame Buffer** :描画結果を格納します。

### § 3.5 タイル分割処理の概念

PowerVR2 は、画面を 32 ドット×32 ドットのタイルという単位に分割して処理するという風変わりな特徴を備えています。

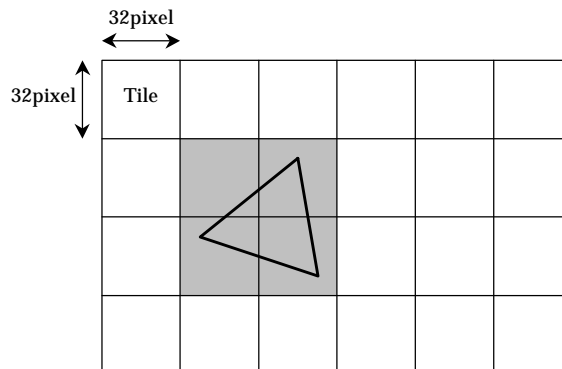


図 3-3 タイル分割小領域

ポリゴンの描画はすべてタイル単位で行われます。

PowerVR2 コアの中には、1 つのタイルを非常に高速にレンダリングするためのモジュールが含まれているので、このチップは他のグラフィックチップに比べて大量のポリゴンを描画することができるのです。

まず、ユーザがポリゴンの表示を要求すると、どのポリゴンがどのタイルに属しているのかをハードウェア (タイルアクセラレータ) が計算し描画エンジンにデータを渡します。

描画エンジンは、タイルごとにポリゴンのソートを行い、陰面処理や透過処理を施し、該当するフレームバッファに書き込みます。

### § 3.6 タイルアクセラレータによる描画のしくみ

タイル分割処理は、タイルアクセラレータと呼ばれるハードウェアによって行われます。タイルアクセラレータは、そのポリゴンを含む矩形領域（長方形）のタイルすべてについてポリゴンが存在していると登録するため、極端な話だと斜めに長い三角形ポリゴンの場合、非常に大きな領域についてそのポリゴンが存在しているとみなします。（図 3-4）

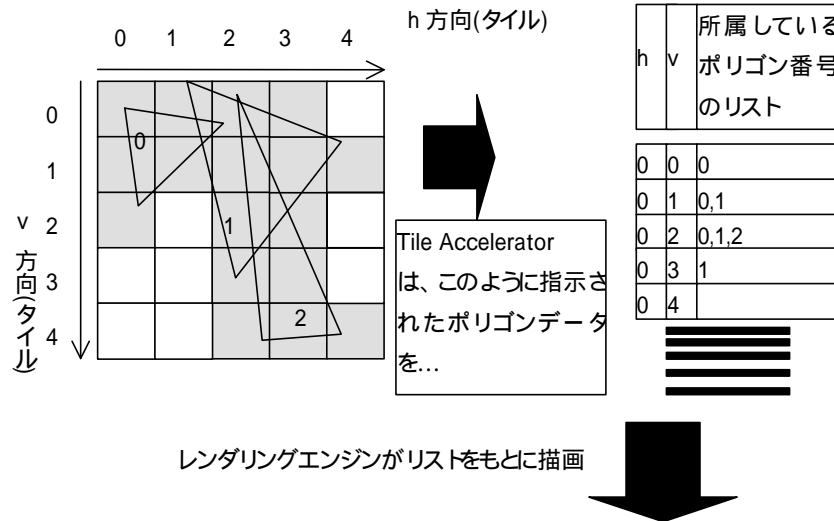


図 3-4 タイル分割処理の概要

登録されているタイルの数が多ければ、結果として何も表示されないとしても多くの処理時間を必要とするので、その分描画の性能が落ちてしまいます。このハードウェアでは斜めに線状に入るようなポリゴンは可能な限り避けたほうがより性能を引き出すことができます。もし、あらかじめこのようになることが分かっていたら、線を小さく分けて表示させたほうが、ポリゴン数は多くなりますが、描画能力は向上します。

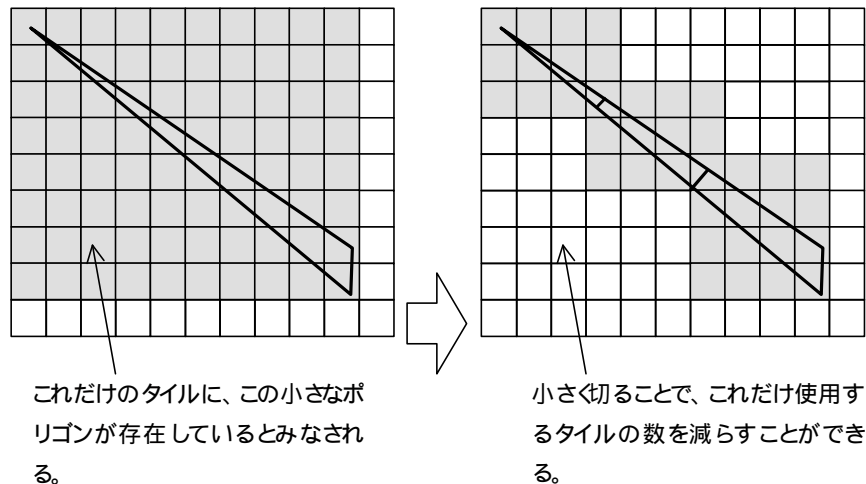


図 3-5 タイル分割による特徴

もし次の描画タイミング（例えば V-Blank In）までに描画処理が終了しない場合、未処理のタイルについては、その前のフレームのデータ（ダブルフレームバッファであれば 2 つ前のフレーム）の絵が残ってしまいます。

SATURN を含む一般的なレンダリング方法では、ポリゴンごとに描画が落ちるため、ポリゴンがところどころ消えたようになります。

また、どのタイルから処理するのかをユーザ側で指定することができるため、極端な話をいえば、中央から螺旋上に描画するようなこともできます。



さらに、描画落ちをしたときに画面への影響を極力小さくしたり、普段は変化しない部分の描画を後に回すなどの技法を用いることもできます。

注釈)PowerVR アーキテクチャについては、NEC 社のホームページにて解説されていますのでそちらもご利用下さい。

## § 3.7 ポリゴンリスト

Dreamcast では、次の 5 種類のポリゴンリストを作成し、Vertex Buffer に格納します。

- |                                      |   |
|--------------------------------------|---|
| ・ <b>Opaque</b>                      | : オペイク (不透明)                                  |
| ・ <b>Opaque Modifier Volume</b>      | : オペイク・モディファイアボリューム<br>(不透明モディファイアボリューム)      |
| ・ <b>Translucent</b>                 | : トランスルーセント (半透明)                             |
| ・ <b>Translucent Modifier Volume</b> | : トランスルーセント・モディファイアボリューム<br>(半透明モディファイアボリューム) |
| ・ <b>Punch Through</b>               | : パンチスルー (抜きテクスチャ)                            |

Opaque は、後ろの絵が透けて見えないポリゴンで、非常に高速に処理することが可能です。  
一方 Translucent は、そのポリゴンを通して、後ろのポリゴンや背景が透けて見えます。  
たとえば、色付きのガラスのような効果を出すこともできますが、ある部分を完全に抜いてしまうこともできます。

ただし、Translucent は、Opaque のように高速に処理できないため、多くの Translucent が重なってしまうと極端に処理スピードが落ちてしまいます。

つまり極力 Opaque を使用し、Translucent を減らすことで、非常に高い描画能力を発揮することができるのです。

Punch Through は、ポリゴンの中に完全な透明色が含まれるもので、主に木や文字、スプライト処理等に用いられます。

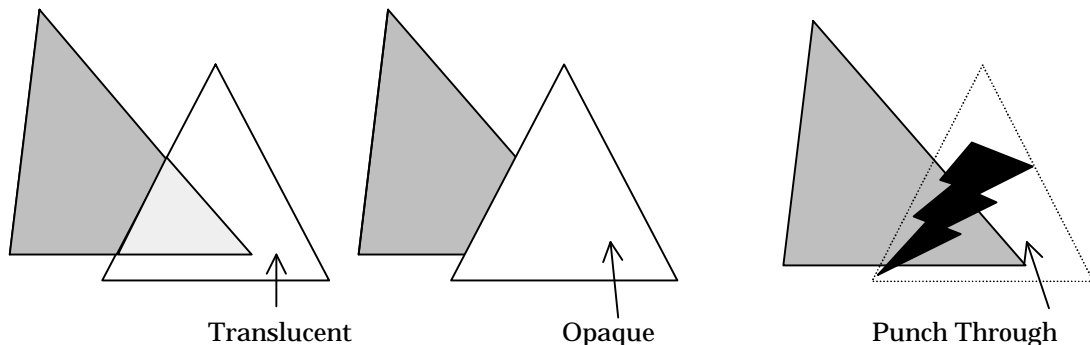


図 3-6 ポリゴンリストタイプ

## § 3.8 ストリップ

ポリゴン形状として、独立三角形ポリゴン、および独立四角形ポリゴンをサポートしています。

それ以外に、ポリゴン処理の際の頂点計算を高速化する手段として、OpenGL など知られた Triangle Strip という技法を使用できます。

Strip を使用すると、ポリゴンのデータ量を削減するだけでなく計算時間を短縮できます。

ストリップの長さとしては、無限長のストリップデータにも対応しています。

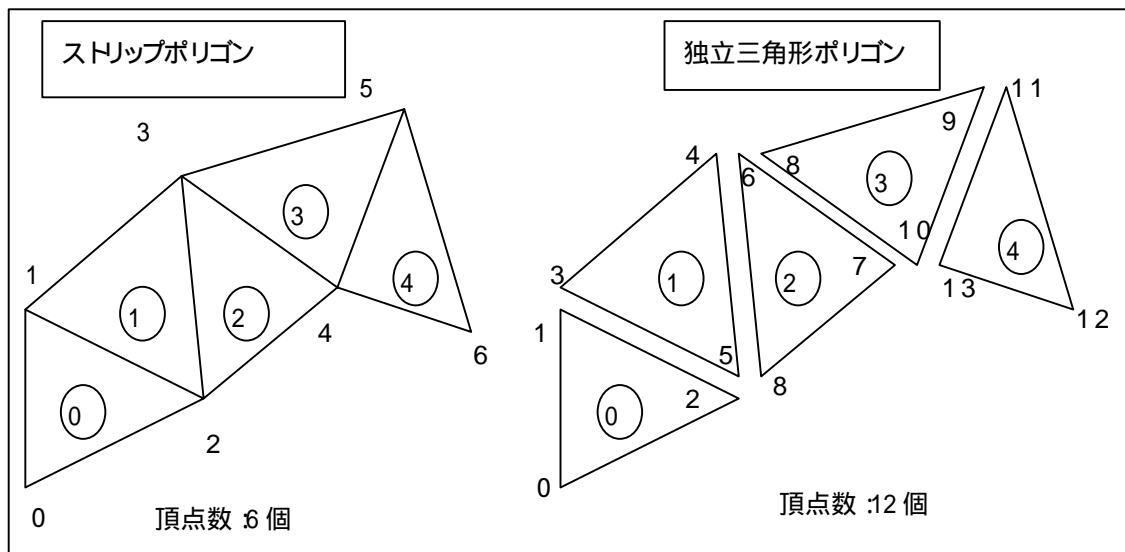


図 3-7 ストリップ系ポリゴンと独立系ポリゴンの違い

例えば、ポリゴン1は ストリップポリゴンの場合は頂点番号 1,2,3 で構成されるのに対して、独立三角形ポリゴンの場合は頂点番号 3,4,5 で構成されます。ストリップポリゴンでは、前のポリゴンのデータを作成したときに使ったデータを流用するので、その分計算をする手間が省けます。しかしながら、もし前のポリゴンが隠れて見えない場合でも、前のポリゴンのデータを必要とするような構造なので、常にストリップ単位で計算しなければなりません。

ストリップを使用するときは、このような無駄がない構造にしなければなりません。

また、上图 3-7 のモデルでは、ストリップのモデルは、全部で 6 頂点で構成されるのに対して、独立三角形のモデルでは倍の 12 頂点で構成されます。ストリップポリゴンは必要なデータ量が少ない構造でもあるのが分かります。

### § 3.9 SortMode

Translucent の場合、ポリゴンのソートを自動的に行う機能があります。

ポリゴンのソートを自動的に行うと、ポリゴン同士が重なった場合でも正しい描画結果を出力できますが、重なり数だけの比較処理が入るため、時間がかかります。

あらかじめソフトでソートしてからハードウェアにデータを渡す Pre Sort モードもサポートされていますが、自動的にソートを行ったのとは異なり、交差したポリゴンの半透明結果が正しくありません。

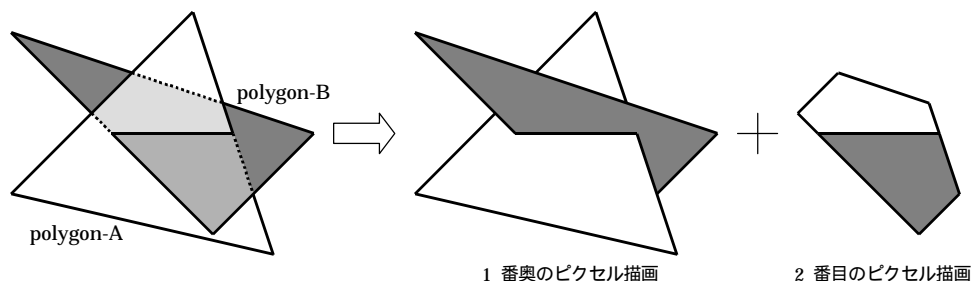


図 3-8 Auto Sort による結果

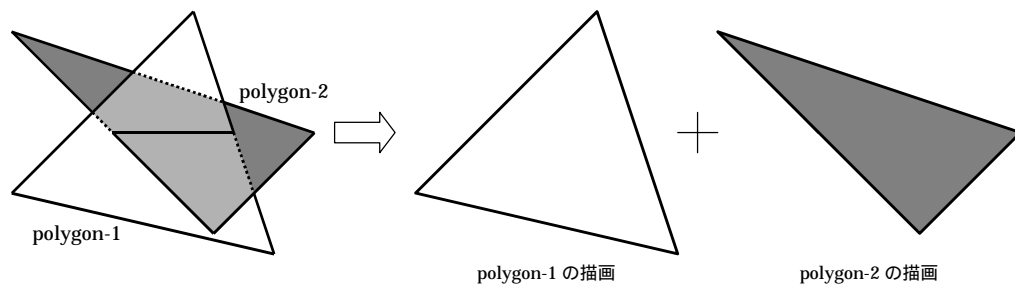


図 3-9 Pre Sort による結果

### § 3.10 頂点カラー

Dreamcast のハードウェアは、頂点に対してさまざまな効果を与えることができます。

たとえば、各頂点ごとに別々の色を与えて、その間にあるポリゴンの色属性を変化させる グーローシェーディング (色補完シェーディング) や、光が当たって明るくなっている頂点の色の輝度を上げて、ハイライトがかかっているように見せる スペキュラー効果 などです。

Dreamcast には、各頂点が元々持っている色情報や、それに加えて与えることができる色情報 (カラーオフセット) を指定することもできるので、ポリゴンで作られたモデルが光の加減や位置によって変化するようなリアルな映像を作り出すことが可能です。

### § 3.11 Modifier Volume (モディファイアボリューム)

Modifier Volume とは、あるモデルに対して別の仮想的なモデルを与え、仮想モデルがそのモデルと交わっている部分とそうでない部分とでテクスチャやマテリアルなどのデータを変えてしまう機能のことです。下の (図 3-10) を見てください。

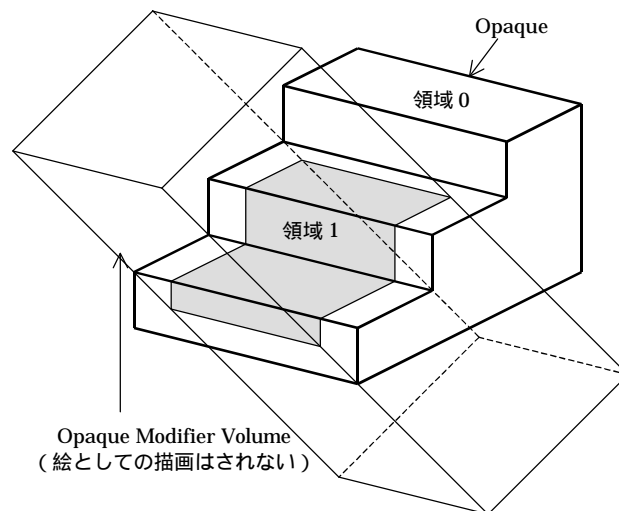


図 3-10 Modifier Volume の原理

(図 3-10) では、モデルが重なったところのテクスチャデータには、仮想モデルが持っていたテクスチャデータが貼られますが、逆に重なったところだけ、元のモデルが持っていたテクスチャデータで、重なっていないところを仮想モデルのテクスチャデータにすることもできます。

前者を Inclusion Modifier Volume、後者を Exclusion Modifier Volume といいます。

また、差し替えるものはテクスチャである必要はなく、マテリアルのデータでも頂点の色でも構いません。

この Modifier Volume は図を見ても分かるように、凹凸のあるモデルに影をつけるときに有効な方法です。ただ影の場合は、単純に影の部分だけを暗くするだけなので、別のテクスチャやマテリアルを設定するのは少々無駄が多いため、重なった部分または重なっていない部分だけを単純に暗くする Cheap Shadow モードも用意されています。

これらのモードの違いを に示します。

表 3-1 Modifier Mode

	Parameter Section Mode	Cheap Shadow Mode
特徴	領域内には別のテクスチャを貼ったり、色を自由に変えたりできる。	領域を暗く変えるだけ。
長所	影の部分のテクスチャを変えるなどのさまざまな表現が可能。	影をつけるという決まりきった処理の場合、リソースの消費が少ない分効率がよい。影にはこれで十分。
短所	普通のモデルに比べて 2 倍のリソースを必要とする。	影以外の表現は無理

Modifier Volume を使う際には注意しなければいけないことがあります。

それは、半透明モデルには半透明の仮想モデルを、不透明モデルには不透明の仮想モデルを使用しなくてはならないということです。

半透明モデルに、不透明の仮想モデルをあてた場合、たとえ物理的に交わっていたとしても、Modifier Volume の処理は起こりません。

簡単な図で説明しましょう。

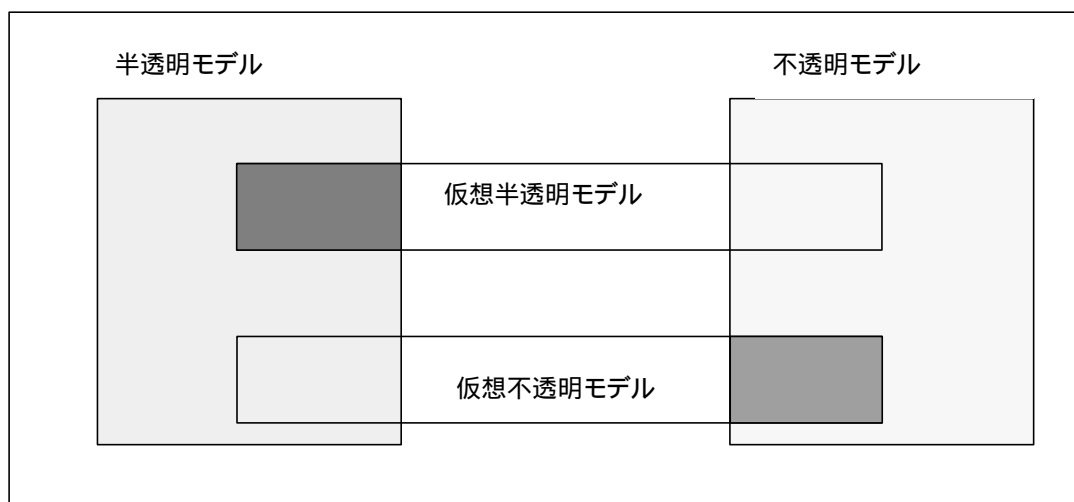


図 3-11 Modifier Volume の注意点

(図 3-11) では、半透明モデルと仮想半透明モデルの間では模様が変わっていますが、不透明モデルと仮想半透明モデルの間は交わっているにもかかわらず模様 (テクスチャ) は変わりません。同様に、半透明モデルと仮想不透明モデルの間にも同じことが起こっています。

このため Vertex Buffer には、それぞれ専用のポリゴンリストを格納します。

## § 3.12 MipMap

Z 方向に動きのある画面では、テクスチャデータを拡大、縮小する必要があります。今までのゲーム機では単に一枚のテクスチャを拡大縮小していたため、拡大時には絵が荒く、縮小時には絵がつぶれてジラジラしました(図 3-12)。この場合 Z 方向の距離に応じて、あらかじめフィルタリングしてあるテクスチャを用意し差し替えることで、視覚的に自然でなめらかな表現が可能になります。この方法を MipMap ( ミプマップ ) と呼びます。

MipMap 処理を行うためには、テクスチャのサイズは、指定サイズから  $1 \times 1$  の最小サイズまでは 2 の累乗になるようにデータを作成します。(図 3-13)

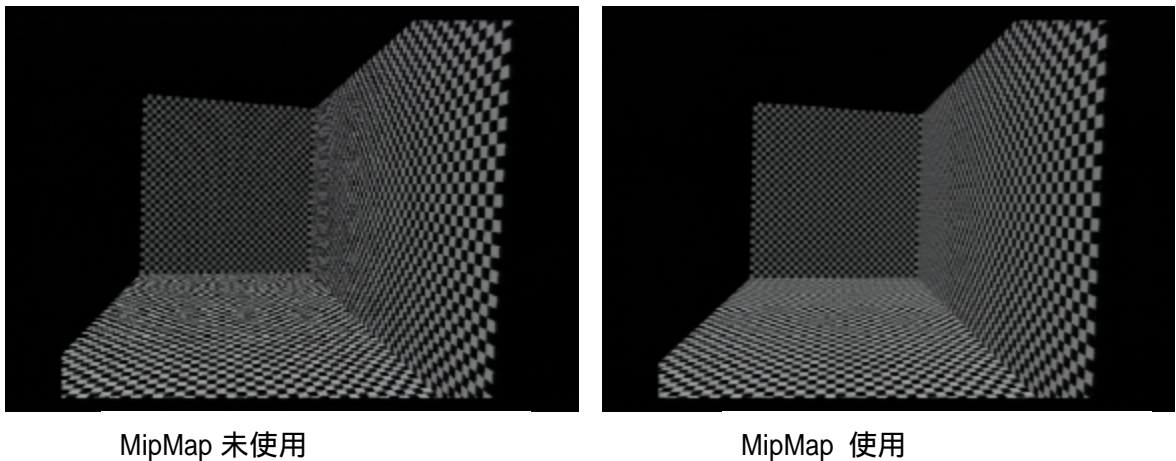


図 3-12 MipMap 効果

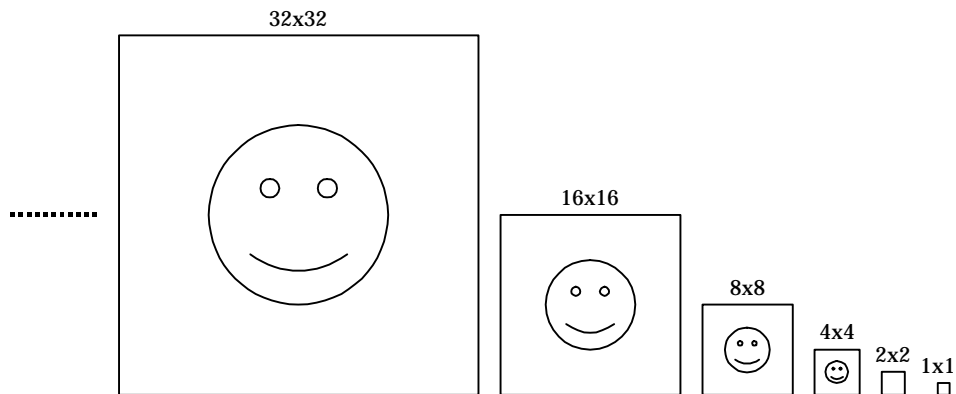


図 3-13 MipMap テクスチャ

### § 3.13 アルファブレンディング(混合処理)

Dreamcast の混合処理は、OpenGL、Direct 3 D と等価で、単純な半透明処理以外に、光学フィルタを通した様な画像を作成したり、レンズフレア、煙、遠くにあるオブジェクトを背景に溶かし込むデプスキューイング処理を行うことも可能です。アルファブレンディングを行うポリゴンは、Translucent のポリゴンリストに登録する必要があります。そのため、Vertex Buffer に専用の領域を確保する必要があります。

アルファブレンディング処理は、陰面のテクスチャデータを参照するため描画が重くなります。また、半透明ポリゴンの AutoSort では、重なりが多くなると重なり数分の比較処理が入るので、描画性能が低下します。

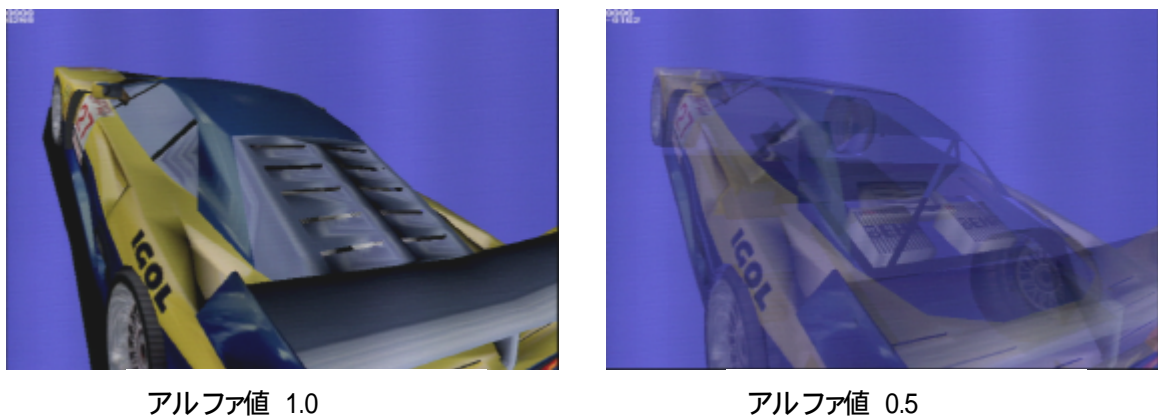


図 3-14 アルファブレンディング

## § 3.14 バンプマッピング

バンプマッピングは、テクスチャデータに、ピクセルデータの代わりに表面の凹凸情報を持たせ、本来は凹凸がないのにもかかわらずあたかも凸凹しているかのように見せる技術です。

本来のポリゴンの上に仮想的な凸凹したポリゴン形状を仮定し、各テクセル上での仮想面の法線ベクトルを数値化して、テクスチャデータとして記述します。

したがって、バンプマッピング用テクスチャのテクセルデータは、カラーデータではなく、そのテクセルの法線ベクトルを2つの角度 S、R で表します。

また、単に凸凹の形状のみが表現できればいいので、この法線ベクトルは常に単位ベクトルとして輝度情報が計算されます。

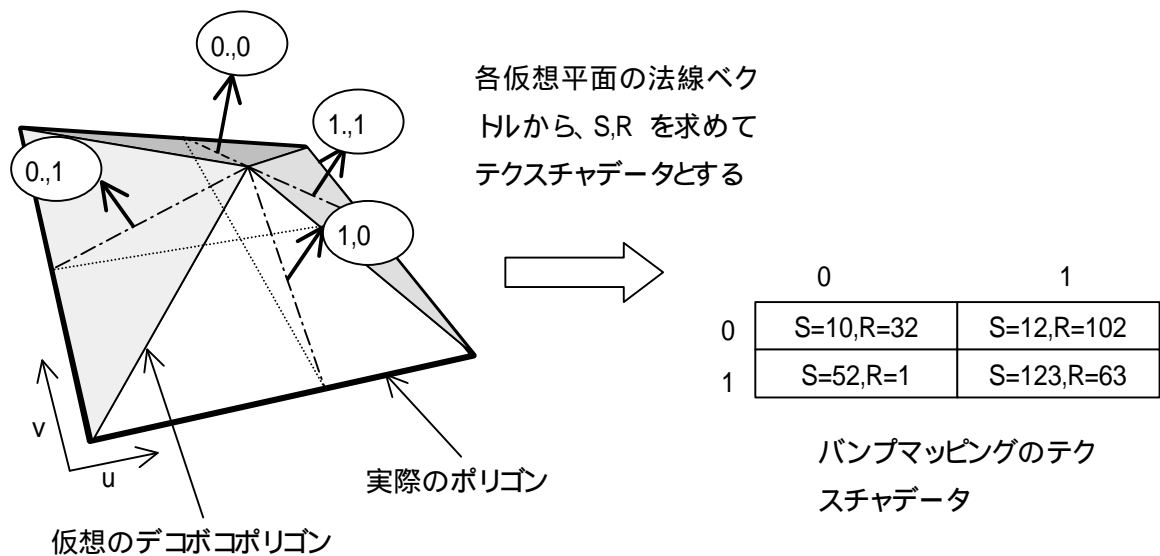
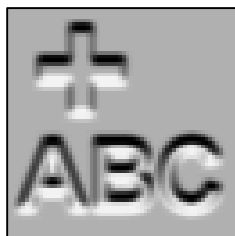


図 3-15 バンプマッピングの原理

**バンプマップテクスチャはバンプマッピング処理されると 値のみが変化する単一色テクスチャとなります。このままではオブジェクトを凸凹に見せることはできませんので、他のテクスチャとあわせて使用します。**

たとえば、テクスチャをデコボコに見せるには、バンプマップ処理されたテクスチャを貼ったポリゴンを描画し、その上に半透明テクスチャを描画します。そうすることで、テクスチャを凸凹に見せることができます。

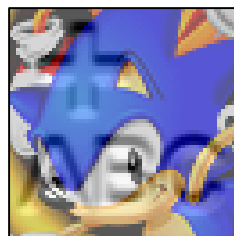
### Bump Mapped ポリゴン



Decal Alpha

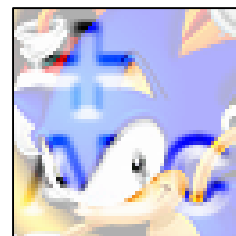
Base Color = 0x0000 0000

### Textured ポリゴン + Bump Mapped ポリゴン



Decal Alpha

Base Color = 0x8000 0000



Modulate Alpha

Base Color = 0xFFFF FFFF

<Blend Function>

SRC = SRC Alpha, DST = Inverse SRC Alpha

図 3-15 テクスチャ + バンプマッピング

## § 3.15 テクスチャフォーマット

### § 3.15.1 ピクセルデータ概略

Dreamcast のテクスチャ用のピクセルデータ (テクセル) として使用できるフォーマットは次のとおりです。

表 3-2 テクスチャのピクセルフォーマット

形式		詳細
RGB	1555	1 ビット、RGB 各 5 ビット
	565	R5 ビット、G6 ビット、B5 ビット
	4444	RGB 各 4 ビット
YUV		YUV 各 8 ビット
Bump Mapping		SR 各 8 ビット
Palette	4bit	各パレット 16 色
	8bit	各パレット 256 色

パレットは1024 色まで持てます。

### § 3.15.2 RGB テクスチャ

RGB テクスチャは 1 テクセル当たり 16bit で示すもので、3 種類のカラーフォーマットがあります。

#### RGB1555 Texture

Bit 15	14-10	9-5	4-0
Alpha	Red	Green	Blue

#### RGB565 Texture

bit 15-11	10-5	4-0
Red	Green	Blue

#### RGB4444 Texture

bit 15-12	11-8	7-4	3-0
Alpha	Red	Green	Blue

### § 3.15.3 YUV テクスチャ

YUV テクスチャは 1 テクセル当たりを 16bit で示すもので、横方向に 2 テクセル分で 1 つのデータ (YUV422 データ) になります。左側テクセルの Y データと 2 テクセル分の U データを合わせたものを Y0V データと呼び、右側テクセルの Y データと 2 テクセル分の V データを合わせたものを Y1V データと呼びます。各 Y/U/V データは、どれも 0 ~ 255 の符号なし 8bit 値で指定します。

MPEG データ(YUV420 データ) は、タイルアクセラレータ内のYUV-data Converter を通すことにより YUV テクスチャデータに変換されます。

#### Y0U-data

bit 15-8	7-0
Y0	U

#### Y1V-data

bit 15-8	7-0
Y1	V

YUV テクスチャデータは、CORE 内で次式によって RGB 値に変換されて描画されます。なお、計算結果の RGB 値は 0 ~ 255 でクランプされます。

### § 3.15.4 Bump Map テクスチャ

Bump Map テクスチャは 1 テクセル当たり 16bit で示すもので、テクセル毎の法線ベクトルを表すための 2 つの 8bit パラメータを指定します。

**Bump Map Texture**

Bit 15-8	7-0
S	R

指定する 8bit パラメータの S と R には、下図のような半球上の点へのベクトルを示すための 2 つの角度をセットします。

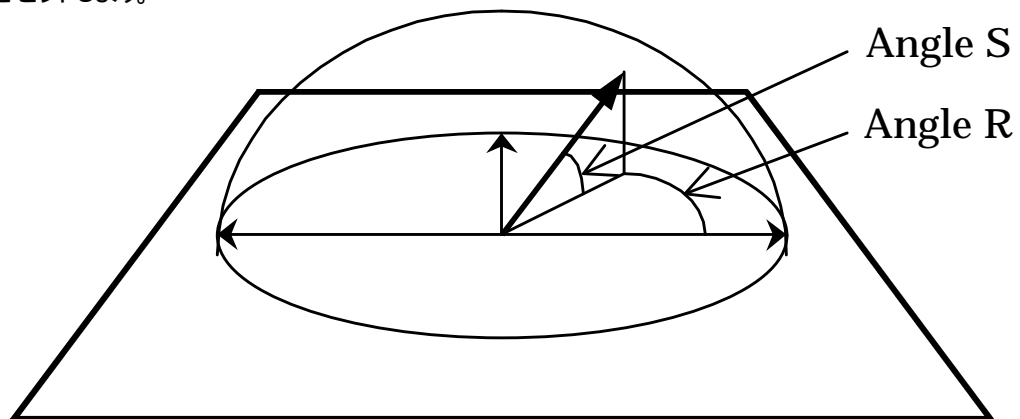


図 3-16 バンプマッピング

このときの半球上の点(x, y, z)は、次のような式で表されます。

$$\begin{aligned}
 x &= \cos(s') * \cos(r') & s' &= \frac{S}{256} \\
 y &= \sin(s') & \text{ただし} & \\
 z &= \cos(s') * \sin(r') & r' &= \frac{R}{256}
 \end{aligned}$$

つまり法線ベクトルを表す角度について、S は 0° ~ 90° を 0 ~ 255 で、R は 0° ~ 360° を 0 ~ 255 で指定します。255 を指定した場合は、256 (すなわち 90° または 360°) とみなされます。

### § 3.15.5 Palette テクスチャ

Palette テクスチャは 1 ピクセル当たり 4bit または 8bit (以下、それぞれ 4BPP および 8BPP) で、COE 内部のパレット RAM の下位アドレスを示します。レジスタにより指定された値を上位アドレスとして付加したものが、パレット RAM アドレス値となります。8BPP の場合は、Palette Selector の上位 2bit のみ有効となります。パレット RAM には 1,024 色まで設定することが可能です。

**4BPP Palette Texture**

bit 9-4	3-0
Palette Selector	Texture Data

**8BPP Palette Texture**

bit 9-8	7-0
Palette Selector	Texture Data



パレット RAM に指定できるカラーデータフォーマットには下表のような 4 種類があり、画面単位で 4 種類の中から 1 つだけ指定できます。複数のカラーデータフォーマットを混在させることはできません。ARGB8888 の場合に、ポイントサンプリング以外のテクスチャフィルタにすると、描画性能が通常の約 1/2 に低下するので注意が必要です。

フォーマット	説明
ARGB1555	値 1bit、RGB 値各 5bit
RGB565	値なし、R 値 5bit、G 値 6bit、B 値 5bit
ARGB4444	値 4bit、RGB 値各 4bit
ARGB8888	値 8bit、RGB 値各 8bit

#### ARGB1555 Palette

bit 15	14-10	9-5	4-0
Alpha	Red	Green	Blue

#### RGB565 Palette

Bit 15-11	10-5	4-0
Red	Green	Blue

#### ARGB4444 Palette

bit 15-12	11-8	7-4	3-0
Alpha	Red	Green	Blue

#### ARGB8888 Palette

bit 31-24	23-16	15-8	7-0
Alpha	Red	Green	Blue

### § 3.15.6 テクスチャデータサイズ

テクスチャデータのサイズは非常に限定されており、基本的には  
つまり、8、16、32、...、512、1024 ということになります。

基本的には、縦方向と横方向で同じサイズにする必要はありませんが、**MipMap 時はかならず正方形でなければなりません。またこの場合、最大テクスチャサイズが12 になります。**

通常テクスチャは **Twiddled 形式** という形式を使います。MipMap、VQ 圧縮 の場合にも、この形式を使います。この他に、Non Twiddled 形式があります。Non Twiddled 形式は、正方形、長方形、の他に Stride が対応しております

Stride のテクスチャの場合、テクスチャサイズは

横方向が 32、64、96、128、192、...、512  
縦方向が 32、64、96、128、192、...、992、1024

になります。ただし、縦方向は全画面ですべて同じサイズになります。(Twiddled 形式の場合はこのような制限はありません。)

### § 3.15.7 テクスチャデータ形式

テクスチャのフォーマットは一般的に Twiddled 形式と呼ばれる変則的な形式が取られます。

Twiddled 形式は、テクスチャデータへの高速なアクセスをするために作られたフォーマットです。それ以外に、一般的なスキャンラインフォーマットもサポートしています。下の表に Dreamcast が持つテクスチャデータの形式の比較を示します。

表 3-3 Twiddled 形式と NON-Twiddled 形式

	TWIDDLED 形式	NON-TWIDDLED 形式
形式		<p>例 <math>U=16\text{pixel}, V=8\text{pixel}</math> の場合</p>
サイズ	$2^n \times 2^m, 2^n \times 2^m$	$2^n \times 2^m, 2^n \times 2^m, 32 \cdot k \times 32 \cdot l$
圧縮	可能	不可能
特徴	Bi-Linear、Tri-Linear フィルタなどが可能。 MipMap はこのモード。	フレームバッファへの出力をテクスチャ領域に描画した時に使用。 Bi-Linear フィルタは可能
速度	速い	遅い

### § 3.15.8 VQ テクスチャ圧縮

Twiddled 形式を用いた場合、テクスチャを圧縮することができます。

テクスチャの圧縮は VQ (Vector Quantization : ベクトル量子化) と呼ばれる技法を使用します。データの圧縮率やクオリティなどはデータの性質によって変わります。

圧縮の仕方を説明するために、例として図 3-17 のような 8×8 のモノクロのテクスチャを圧縮する場合を考えてみましょう。

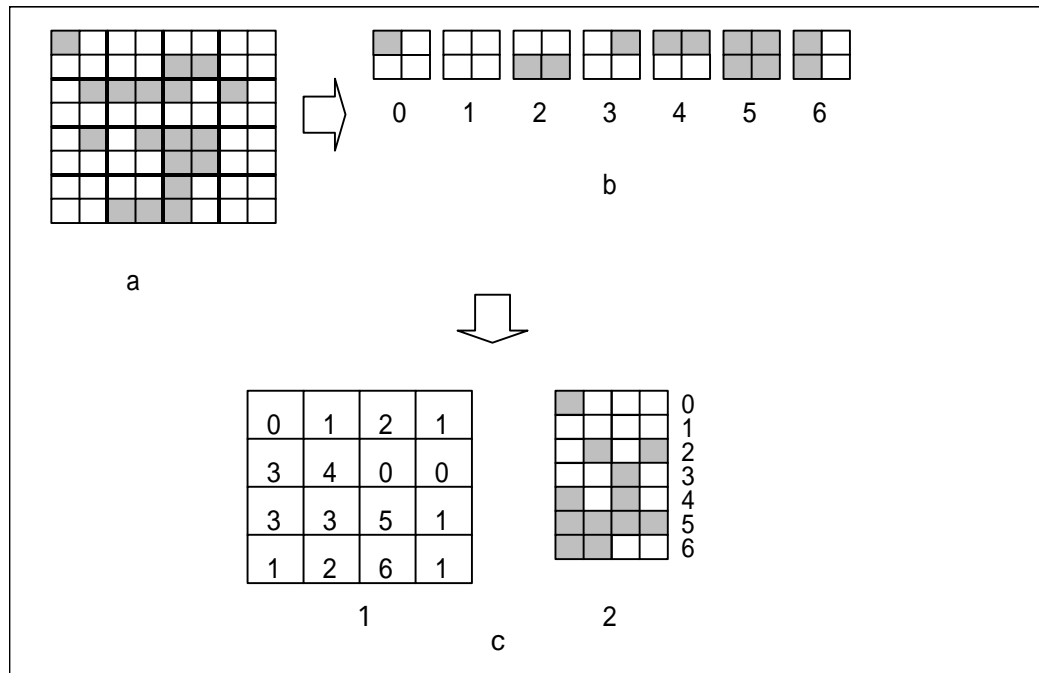


図 3-17 VQ 圧縮の原理

まず、a は元絵です。この元絵をわかりやすくするために、2×2 の格子で切ってみると、この絵は b のように 7 つの 2×2 格子のパターンでできているのが分かります。

つまり a の絵は c のように、どの 2×2 の格子がどのような配列で並んでいるのかという情報に整理できます。これが VQ 圧縮の原理になります。c1 を Index 情報、c2 を Code Book と呼びます。

Dreamcast で使われる VQ は、1 つのテクスチャに必ず 256 個の Code Book を必要とします。

Code Book には 16 ビットのカラー形式で収められるので、VQ 圧縮された 1 枚のテクスチャデータには最低でも、2048 バイト必要とします。

一方 Index 情報は Code Book が 1 つにつき 4 バイトです。つまり 4x2 バイトを 1 バイトにする圧縮方法です。ただし、前述のように、2048 バイト未満のテクスチャを圧縮すると、Code Book の分だけ逆にデータサイズを必要としてしまいます。つまり、32x32 以下のテクスチャデータでは逆にデータサイズが大きくなってしまいます。

さて、VQ 圧縮方式は、8x8 から 1024x1024 までのサイズに対応していますが、実際の使用に耐えられるサイズは、大体 64x64 から 256x256 程度までになるでしょう（それ以上のサイズはテクスチャ自体が大きくて使いづらいでしょう）。VQ 圧縮を有効に使用するには、テクスチャのサイズに気を配る必要があります。

表 3-4 Dreamcast の VQ 方式の特徴

Code Book のサイズ	256 個(2048 バイト)
1 ピクセルのデータサイズ	16 ビット(2 バイト)
使用可能なサイズ	8x8 ~ 1024x1024 32x32 以下はかえって効率が落ちる。 (Small VQ を使用することで解決)
使用できるフォーマット	Twiddled

## § 3.16 テクスチャフィルタ

テクスチャフィルタには、以下の 4 つがあります。

- ・ **Point Sampling**
- ・ **Bi-Linear Filter**
- ・ **Tri-Linear Filter**
- ・ **Texture Super Sampling**

これらフィルタリングの特性を以下の表に示します。

表 3-5 テクスチャフィルタの種類

	Point Sampling	Bi-Linear Filter	Tri-Linear Filter	Texture Super Sampling
特徴	テクスチャデータからピクセルデータへの変換が 1 対 1。通常のテクスチャマッピング処理。	対象になるテクセルデータの近傍 4 テクセルからピクセルデータを算出。	MipMap を使っているときだけ。前後の MipMap に Bi-Linear Filter をかけて加重平均をとる。	4 倍のサンプリングポイントを使い、より高精度なフィルタリングをする。
長所	最も処理が軽い。	拡大・縮小時のちらつきなどが軽減。	MipMap テクスチャの切り替わりがスムーズ。	他の 3 つのフィルタリングとあわせて使うことができる。非常に精度が高い。
短所	縮小したときにテクスチャの絵が崩れる。	絵がぼけたようになる。Non-Twiddled テクスチャの場合、Point Sample の 2 倍の処理時間がかかる。	非常に計算時間がかかる。	通常の 3 ~ 4 倍の時間を必要とする。

### § 3.16.1 Point Sampling(ポイントサンプリング)

ポイントサンプリングは、サンプリングポイント(x, y)から計算されたテクスチャ座標(u, v)のデータを描画ピクセルのテクスチャデータとします。

テクスチャマッピング処理の中で最も処理が軽いモードですが、拡大縮小した際の絵のクオリティは低くなります。

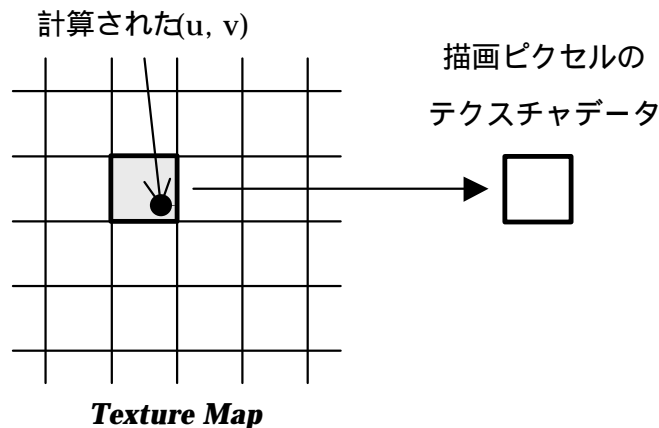


図 3-18 Point Sampling

### § 3.16.2 Bi-Linear Filter (バイリニアフィルタ)

バイリニアフィルタは サンプリングポイント(x, y)から計算されたテクスチャ座標(u, v)のデータとその周り 3 つの計 4texel 分のデータを加重平均して、描画ピクセルのテクスチャデータとします。

4texel 分のデータから加重平均するので、拡大縮小した際の絵のクオリティはポイントサンプリングより高くなります(場合によっては、絵がボケているように見える)

**Twiddle 形式**テクスチャの場合はポイントサンプリングと比べても処理時間はほとんど変わりませんが、**Non-Twiddle 形式**テクスチャの場合はワーストケースで 2 倍の処理時間がかかる可能性があります。

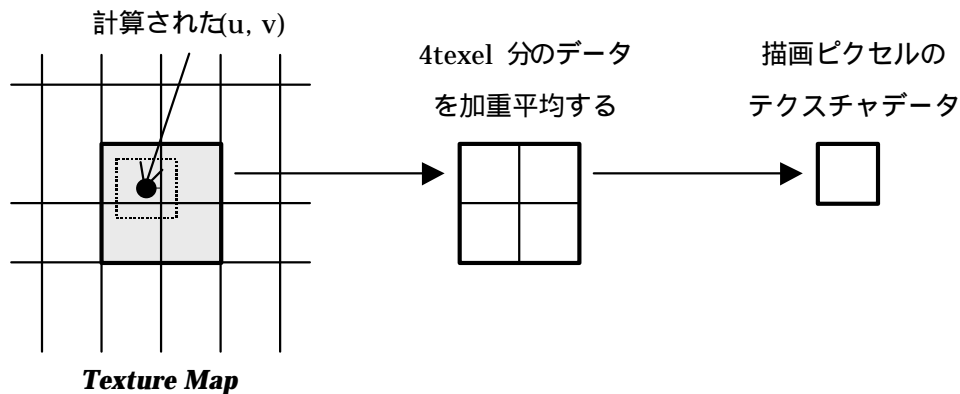


図 3-19 Bi-Linear Filter

バイリニアフィルタリングは 4texel 分のデータを使用するので、サンプリングポイントから計算されたテクスチャ座標がテクスチャマップ端の場合、その隣にある texel すなわち、反対側のテクスチャマップ端にある texel が使用されて、予想外の色のピクセルが表示されてしまうことがあります。**(テクスチャ UV の Clamp 機能を使用すれば回避可能)** また、大きなテクスチャマップから一部分を切り出して使用する場合にも同様なので、注意が必要です。

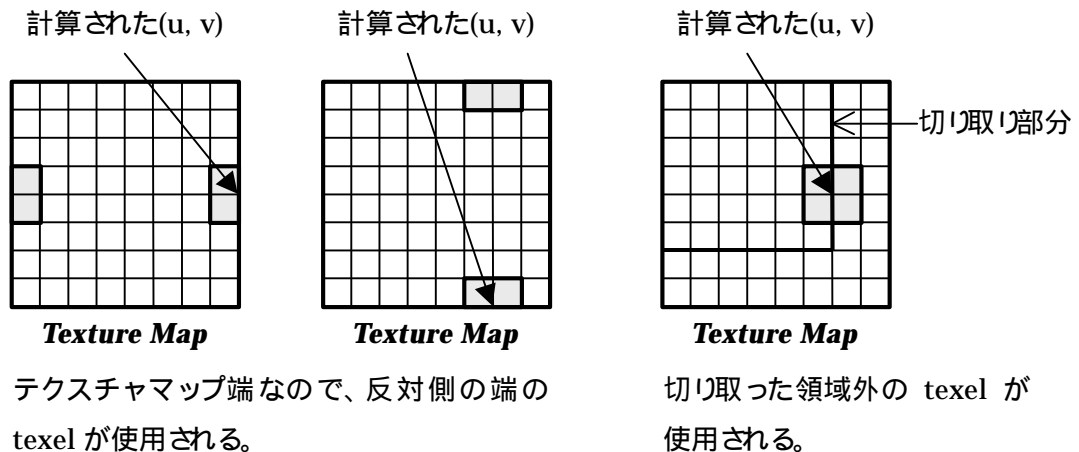


図 3-20 UV 設定によるフィルタリングの影響

透明 texel のあるテクスチャにバイリニアフィルタ処理を行ったとき、透明 texel データ(値 カラー値どちらも)も 4texel 分の加重平均計算に使用され、描画ピクセルの値はその計算結果になります。Translucent の場合は、計算結果の値を使って描画されます。しかし、**透明ピクセルと不透明ピクセルの境界部分において透明texel のカラーデータが影響してしまうので注意が必要です。**

### § 3.16.3 Tri-Linear Filter(トライリニアフィルタ)

1 つのテクスチャマップを使用している場合は、バイリニアフィルタを行うことでポイントサンプリングよりも絵のクオリティを高めることができます。また、縮小率が大きくなった(Z 方向へ移動した)ときの絵のクオリティを上げるためには、MIPMAP 処理を同時に行います。しかしバイリニアフィルタと MIPMAP 処理を併用しても、異なるサイズの MIPMAP テクスチャの切り替わりがはっきりと見えてしまいます。

トライリニアフィルタは、2 つのサイズの MIPMAP テクスチャに対して、バイリニアフィルタを行った結果を加重平均し、描画ピクセルのテクスチャデータとします。

全部で 8texel 分のデータから加重平均するので、拡大縮小した際の絵のクオリティは最も高くなり、MIPMAP テクスチャの切り替わりもスムーズですが、その分処理時間は最も大きくなるので、描画時間に余裕のあるアプリケーションでの使用を推奨します。

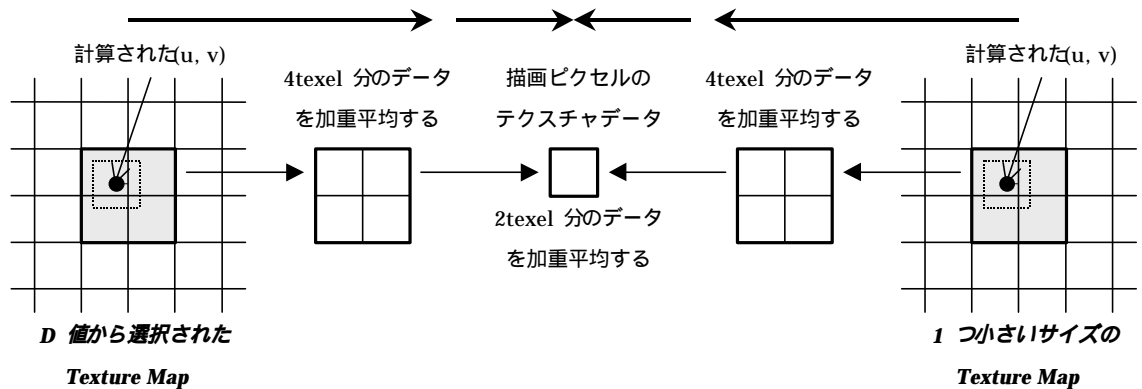


図 3-21 Tri-Linear Filter

トライリニアフィルタもバイリニアフィルタと同様に、計算されたテクスチャ座標によっては予想外の色のピクセルが描画されてしまうことがあるので、注意が必要です。また、透明 texel のあるテクスチャについても同様になります。

トライリニアフィルタリングを施したポリゴン描画は Opaque の場合には 2 回の処理で、Translucent の場合には 3 回の処理で行います。すなわち、Opaque には 2 ポリゴン分、Translucent には 3 ポリゴン分の同一形状のポリゴンリストを確保する必要があります。

### § 3.16.4 Texture Super Sampling (テクスチャスーパーサンプリング)

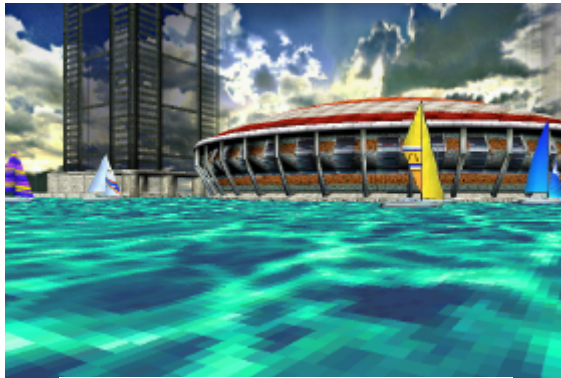
テクスチャスーパーサンプリングは、3 種類の各フィルタリングモードと組み合わせて使用することが可能です。

この機能は一般的には、アンアイソトロピックフィルタと呼ばれるもので、1pixel 当たりのテクスチャサンプリングポイントを  $(x, y)$ ,  $(x+0.5, y)$ ,  $(x, y+0.5)$ ,  $(x+0.5, y+0.5)$  の縦横 2 倍にして、テクスチャを縮小して描画する部分のクオリティを上げる機能です。しかし、読み込むテクスチャデータが 4 倍になるので、この機能を使用しない場合に比べ、3~4 倍の描画時間が必要となります。

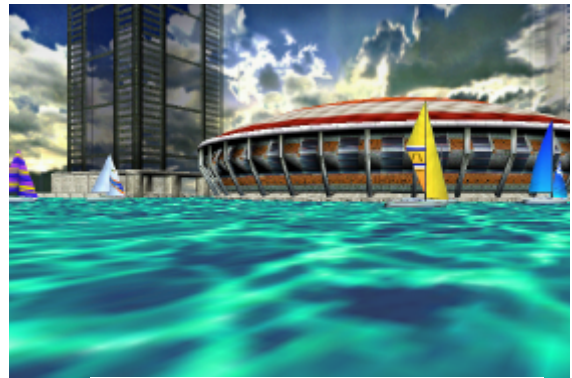
細かな模様や線などのテクスチャを使用したもので、縮小した場合のクオリティを上げたいポリゴンなどにのみ使用することを推奨します。また、Image Super Sampling を使った画面全体のフィルタリングと同時に使用してもあまり効果が見られないため、その際は描画性能の観点から画面全体のフィルタリングのみを行うことを推奨します。

### § 3.16.5 各種テクスチャフィルタの画面効果

各種テクスチャフィルタリングの効果については、下図を参照してください。



Point Sampling



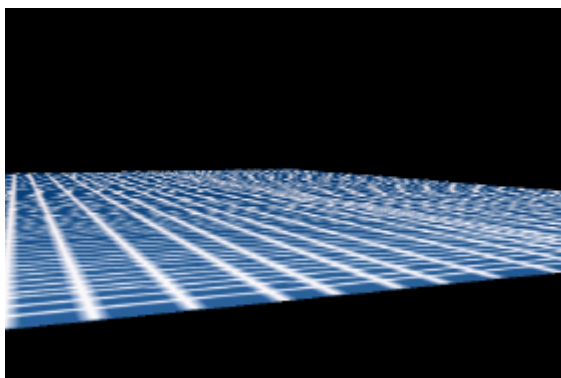
Bi-Linear Filter



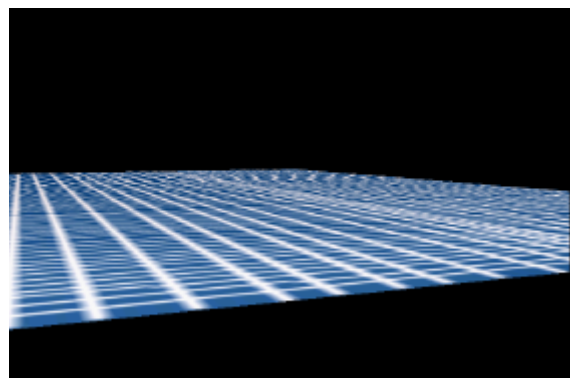
No Tri-Linear Filter



Tri-Linear Filter



No Texture Super Sampling



Texture Super Sampling

図 3-22 各種フィルタによる画面効果

## § 3.17 フォグ

フォグ（霧）の効果を出します。ハードウェアにある程度の計算をさせてしまう方法と、自分で計算して頂点データの一部として設定する方法があります。

ハードウェアに計算させる場合、Z 方向つまり奥行き方向にのみフォグを掛けることができますが、自分で計算する場合は上下方向に霧の効果を出すことも可能です。

ハードウェアに計算させるモードは Lookup Table Mode と呼ばれ、Z 方向のある単位ごとに霧の掛かり方をテーブル化して、その間の Z 位置についてはハードウェアが補間して、霧の掛かり具合を割り出すという方法です。

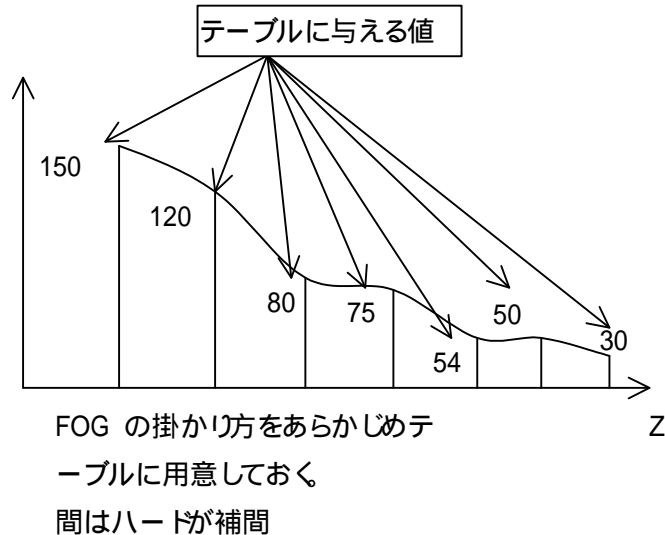


図 3-23 FOG テーブル

一方 Per Vertex Mode は、頂点ごとにすべて霧の掛かり具合を計算し、ハードウェアはレンダリングの処理のみを担当するというもので、Lookup Table Mode のように Z 方向にしか霧が掛からないということはありませんが、すべて自分で計算しなければならないので処理に時間がかかります。

## § 3.18 クリッピング

クリッピングには、タイルアクセラレータで行われる Tile 単位のタイルクリッピングと、フレームバッファへの書き込みの際に行われるピクセル単位のピクセルクリッピングとがあります。

### § 3.18.1 タイルクリッピング

タイルアクセラレータに入力されたポリゴンリストはタイル単位でクリッピングすることができ、指定されたクリッピング領域を完全に外れたポリゴンはテクスチャメモリへ格納されません。タイルアクセラレータでのタイルクリッピング領域は **Global Tile Clip 領域**と User Tile Clip の Control Parameter で指定される **User Tile Clip 領域**で決まります。Global Tile Clip は、レジスタ指定なので画面単位でしか指定できません。User Tile Clip は「使用しない」「内側有効」「外側有効」の中から Object 単位に選択でき、その領域サイズも Object 単位に設定が可能です。



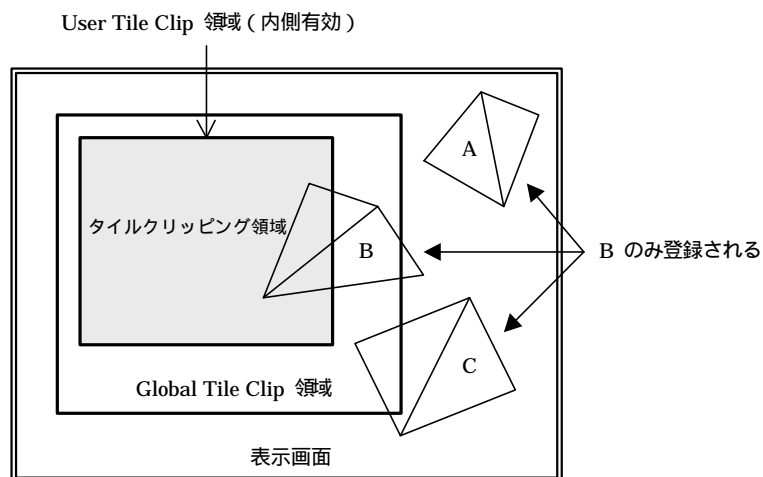


図 3-24 タイルクリッピング

例えば、下図のように 1 画面の中にタイル単位で 2 つに分割された画面を描画させる場合は、それぞれのエリアのポリゴンデータを、タイルアクセラレータに入力する際にタイルクリッピングを行います。

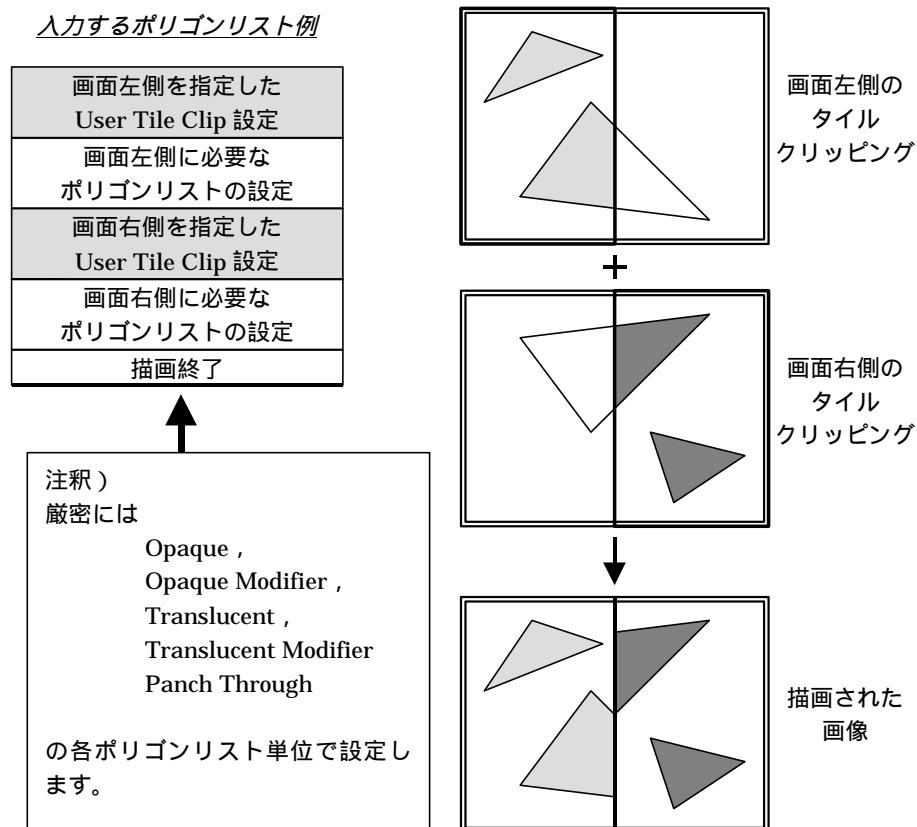


図 3-25 画面分割ゲームの用例(1)

### § 3.18.2 ピクセルクリッピング

CORE 内の**アキュムレーションバッファ**からテクスチャメモリ内のフレームバッファへ転送されるピクセルデータは、ピクセル単位でクリッピングすることができ、指定されたクリッピング領域外側のデータはフレームバッファへ格納されません(ポリゴン描画は行われます)。ピクセルクリッピング領域は、1画面に1つしか指定できません。また、位置のピクセルは領域内側となり、フレームバッファへ格納されます。

表示画面サイズがクリッピングサイズよりも大きな場合、クリッピング領域外の画面にはフレームバッファ内に残っているデータがそのまま表示されます。

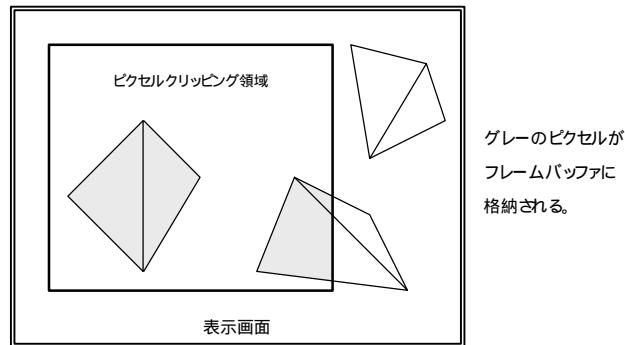


図 3-26 ピクセルクリッピング

例えば、下図のように1画面の中にタイル境界でないウィンドウエリアがあるような画面を描画させる場合は、ピクセルクリッピングを使って1フレームの間に2度の描画を行います。

#### パラメータ入力と描画処理例

##### タイルアクセラレータ初期化

1 回目の描画に必要な Opaque ポリゴン
Opaque のリスト終了
1 回目の描画に必要な Translucent ポリゴン
Translucent のリスト終了

1 回目の  
ポリゴン  
リスト

##### タイルアクセラレータ初期化

2 回目の描画に必要な Opaque ポリゴン
Opaque のリスト終了
2 回目の描画に必要な Translucent ポリゴン
Translucent のリスト終了

2 回目の  
ポリゴン  
リスト

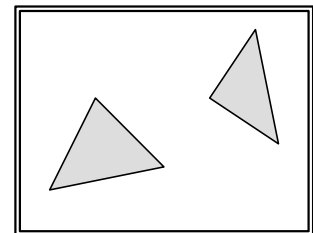
##### ピクセルクリッピング設定 (画面全体)

##### 1 回目のレンダリング

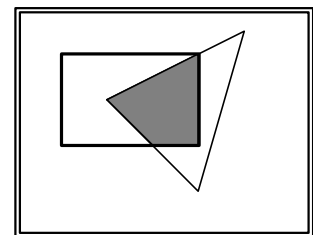
##### ピクセルクリッピング設定 (画面一部)

##### 2 回目のレンダリング

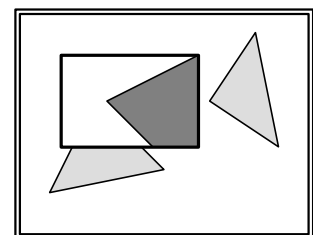
2 回の  
描画処理



1 回目の  
描画



2 回目の描画  
+  
ピクセル  
クリッピング



完成した  
画像

注) 1 回目と 2 回目のディスプレイリストは、別のテクスチャメモリ領域に格納させる。

図 3-27 画面分割ゲームの用例(2)

## § 3.19 バッファ

Dreamcast でバッファと称される部分は、以下の 6 種類です。そのうち、画像処理に関係あるバッファについて説明します。

<b>Vertex Buffer</b>	:ポリゴンリスト情報を格納します。 (CPU によって Work RAM 上に設定されます)
<b>Native Buffer</b>	:PowerVR2 CORE で処理されるディスプレイリスト情報を格納します。
<b>Texture Buffer</b>	:テクスチャデータを格納します。
<b>Accumulation Buffer</b>	:タイル単位の描画結果を格納します。(Holly チップ内部)
<b>Frame Buffer</b>	:1画面の描画結果を格納します。
<b>Strip Buffer</b>	:タイルライン単位の描画結果を格納します。

### § 3.19.1 Native Buffer (ネイティブバッファ)

Native Buffer は、Vertex Buffer で設定されたポリゴンリストを、タイルアクセラレータがタイル分割を行い、その分割されたポリゴンの「頂点情報」「登録タイルのリスト」を格納するバッファです。このバッファに格納されたデータを基に、PowerVR2 コアがタイル単位で Z ソート、描画を行います。この領域は 32Bit でアクセスされます。

Native Buffer は、ダブルバッファ構成をとる必要があり、「登録タイルのリスト」が描画条件により大幅に増減するため、テクスチャバッファの設計には注意が必要になります。

### § 3.19.2 Native Buffer の容量

Native Buffer の容量は

**Native Buffer 容量 = VRAM 全体の容量 - Texture Buffer サイズ + 表示用 Frame Buffer サイズ**

で算出されます。

ただし、Native Buffer (ダブルバッファ) の全容量は、Frame Buffer 全容量の半分を越えることはできません。

すなわちフレームバッファメモリ 8MB の量産システムでは、ネイティブデータバッファの容量は

**最大 2MB × 2 個**

が上限となります。上記の式で計算した結果がこの制限を越えた場合は、実際には制限いっぱいの容量を Native Buffer (ダブルバッファ) として確保し、余った領域はテクスチャ領域として使用されます。

### § 3.19.3 Texture Buffer (テクスチャバッファ)

テクスチャデータを格納するバッファです。テクスチャを格納する領域は、64Bit アクセスになります。このバッファに対しては、テクスチャデータの他に、Accumulation Buffer の結果をコピーすることも可能です。

テクスチャメモリ内でのテクスチャデータの格納状態は、描画性能に大きな影響を与えます。テクスチャメモリは、ハードウェア的に「ページ」と呼ばれる 2048byte ごとのエリアに分割されており、描画性能を低下させないためには、各テクスチャデータをこの 1 ページ内に収めることが重要になります。ちなみに 2048byte というデータ量は、32 × 32 サイズの 16bit テクスチャ 1 つ分に相当します。

様々なサイズのテクスチャを格納する場合は、各サイズを上手に組み合わせてページ境界をまたがないように格納することで、描画性能を最大限に発揮することができます。1 つが 2Kbyte 以上のテクスチャデータも、出来る限りページ境界をまたがないようにすることで、描画性能を低下しないようにすることが可能です。

また、VQ テクスチャの場合は Code Book サイズが 2Kbyte になっているので、Code Book の先頭アドレスを 2Kbyte 境界にすることが効果的です。

#### § 3.19.4 Accumulation Buffer (アキュムレーションバッファ)

PowerVR2 コアに内蔵されている描画用のバッファです。タイル分(32×32)の大きさを持ち、グーロシェーディング、アルファブレンディング等のエフェクト処理を行った結果を格納します。このバッファにある描画結果が、Frame Buffer にコピーされます。

#### § 3.19.5 Frame Buffer (フレームバッファ)

Frame Buffer は、TV 画面 1 枚分の画像データを保存しておき、ハードウェアはそのイメージを参照して、1 枚の TV 画面を作ります。通常、Frame Buffer は、表示用と描画用のダブルバッファ構成となっており、画面が大きければそれだけ多くのメモリを必要とします。Accumulation Buffer の結果がそのまま Frame Buffer にコピーされるだけでなく、Frame Buffer モードの選択により、16Bit カラーに減色したり、32Bit カラーのうちの部分を削ったり、また、カラークランプ機能を使用してフェードイン・アウトの効果を施したり、Image Super Samplingによってアンチエイリアスが掛けられた画像も格納できます。

この領域は32Bit アクセスとなります。

#### § 3.19.6 Frame Buffer サイズの算出方法

Frame Buffer サイズを求める方法を提示します。

表 3-6 フレームバッファのサイズ

画面解像度	カラーサイズ	画面解像度 × カラーサイズ × バッファ数
320x240	16BPP	(320x240) × 2 Byte × 2 Buffer = <b>307200</b> Byte
	24BPP	(320x240) × 3 Byte × 2 Buffer = <b>460800</b> Byte
	32BPP	(320x240) × 4 Byte × 2 Buffer = <b>614400</b> Byte
320x480	16BPP	(320x480) × 2 Byte × 2 Buffer = <b>614400</b> Byte
	24BPP	(320x480) × 3 Byte × 2 Buffer = <b>921600</b> Byte
	32BPP	(320x480) × 4 Byte × 2 Buffer = <b>1228800</b> Byte
640x240	16BPP	(640x240) × 2 Byte × 2 Buffer = <b>614400</b> Byte
	24BPP	(640x240) × 3 Byte × 2 Buffer = <b>921600</b> Byte
	32BPP	(640x240) × 4 Byte × 2 Buffer = <b>1228800</b> Byte
640x480	16BPP	(640x480) × 2 Byte × 2 Buffer = <b>1228800</b> Byte
	24BPP	(640x480) × 3 Byte × 2 Buffer = <b>1843200</b> Byte
	32BPP	(640x480) × 4 Byte × 2 Buffer = <b>2457600</b> Byte

上記条件は、アンチエイリアスをかけても変わりませんが、タイプB のフリッカーフリーを使用した場合にのみ半分になります。

#### § 3.19.7 Strip Buffer (ストリップバッファ)

Strip Buffer は、横方向は画面サイズ分で、縦に短い横長のバッファを2つ使って、走査線の位置にあわせて切り替えて使います。たとえば、走査線が64ライン目から95ライン目までを描画している間はバッファ1を使い、そのときは96ライン目から127ライン目までのレンダリング結果をバッファ2に入れます。96ライン目から127ライン目まではバッファ2を使って描画し、その間にバッファ1に128ライン目から160ライン目までのレンダリング結果をいれます。

Strip Buffer を使用する上で注意しなければいけないのは、Strip Buffer は、そのバッファがTVの描画で使用される時点ですでに準備ができていなければならないということです。

つまり、1つのストリップバッファ分を描画する時間以内に各ストリップバッファ分のレンダリングをしなければならないのです。ポリゴンの量やテクスチャフィルタなど、レンダリング時間はそこに存在するデータの種類によって変化します。それらすべてが1つのストリップバッファ分を描画する時間以内にレンダリングを

終了しなければいけません。

ストリップバッファは 1 個所にポリゴンが集中するようなアプリケーションには不向きです。

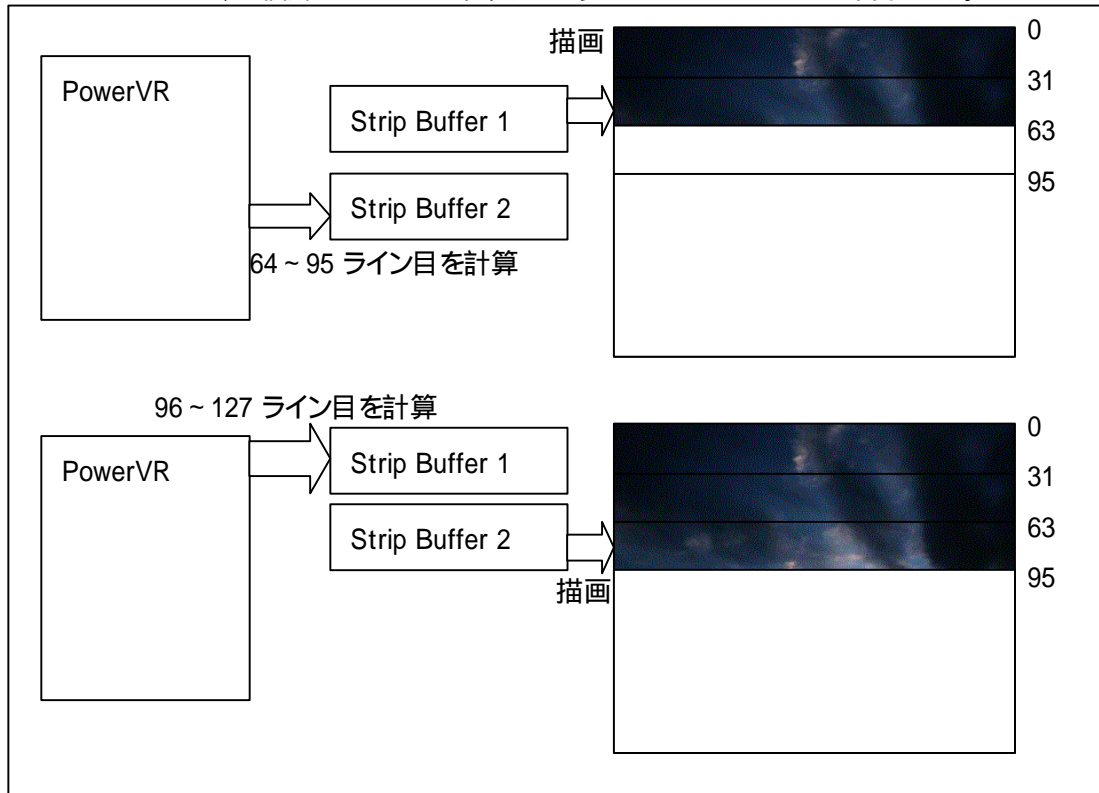


図 3-28 ストリップバッファの原理

### § 3.20 Image Super Sampling

PowerVR2 では フレームバッファイメージに対してフィルタリングを行う機能があります。この機能を利用することで、画面全体に対しアンチエイリアスを行った、インターレース時のフリッカを押さえることが可能になります。この機能は、X 方向2ピクセル、Y 方向3ピクセルを参照して1ピクセルを描画します。

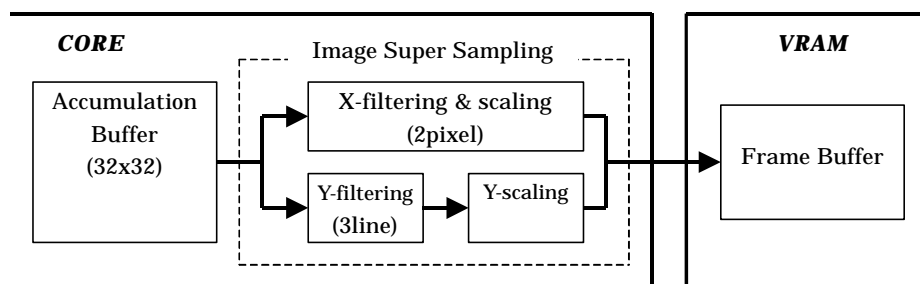


図 3-29 Image Super Sampling の原理

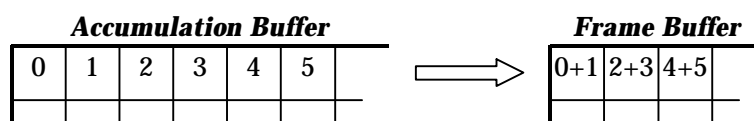


図 3-30 X スケーラ フィルタリング

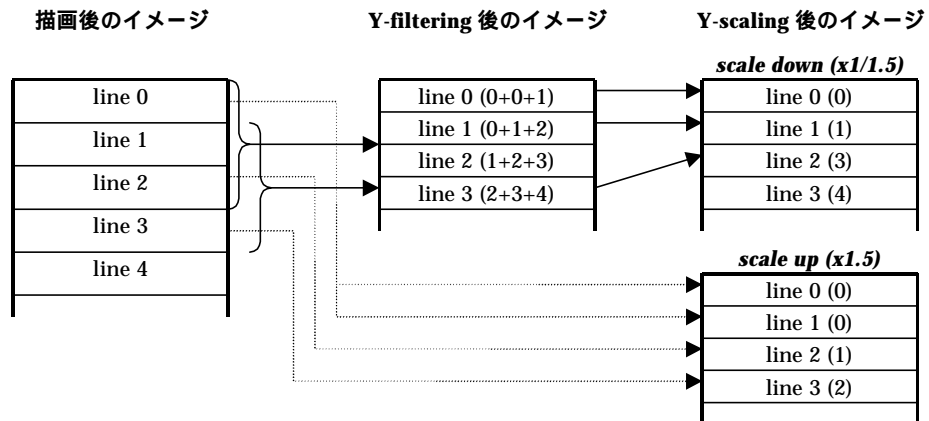


図 3-31 Y スケーラ フィルタリング

## § 3.21 フリッカフリーフィルタリング

上記の Image Super Sampling 機能を使って、インターレス操作によるフリッカを低減することができます。フリッカフリーには 2 種類のモードが存在し、それぞれタイプ A、タイプ B とよびます。

### § 3.21.1 タイプ A

まず Image Super Sampling の Y スケーリング係数をできるだけ 1.0 倍に近い縮小係数にし、Y フィルタリングを有効にします。そして、3 ライン分のデータでフィルタリングを行った結果をフレームバッファへ格納します。つまり、480 ラインで行った描画結果をそのままのライン数でフレームバッファへ格納します。したがって、画面描画は 1 フレーム毎 (1/30 or 1/25 秒単位) に行えば良いことになります。

表示は、フレームバッファからのリード開始アドレスをフィールド毎に 1 ラインずらし、1 ラインずつ飛ばしてリードされます。

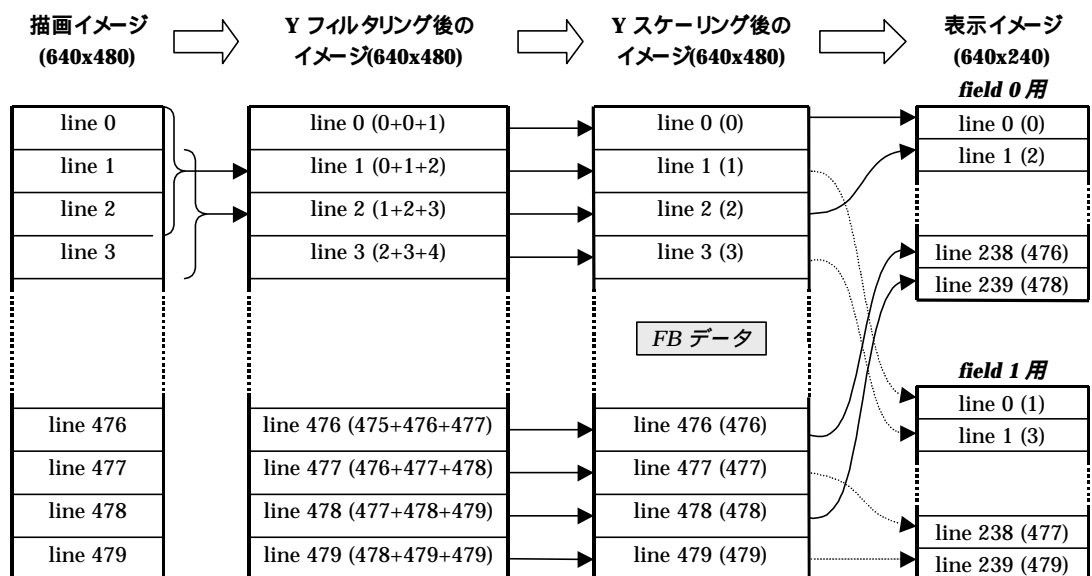


図 3-32 タイプ A フリッカフリー

### § 3.21.2 タイプB

まず Image Super Sampling において 3 ライン分のデータでフィルタリングを行い、その結果を 1/2 スケーリングして指定された表示フィールドのために必要なラインデータだけをフレームバッファへ格納します。つまり、描画は 480 ラインで行う必要がありますが、フレームバッファへ格納されるピクセルデータは 240 ライン分で済みます。

ただし、これらの処理はタイル単位で描画されたピクセルデータをフレームバッファへ転送する際に行われるので、描画は毎フィールド (160 or 1/50 秒単位) 行わなければなりません。

表示は、フレームバッファからのそのままリードされます。

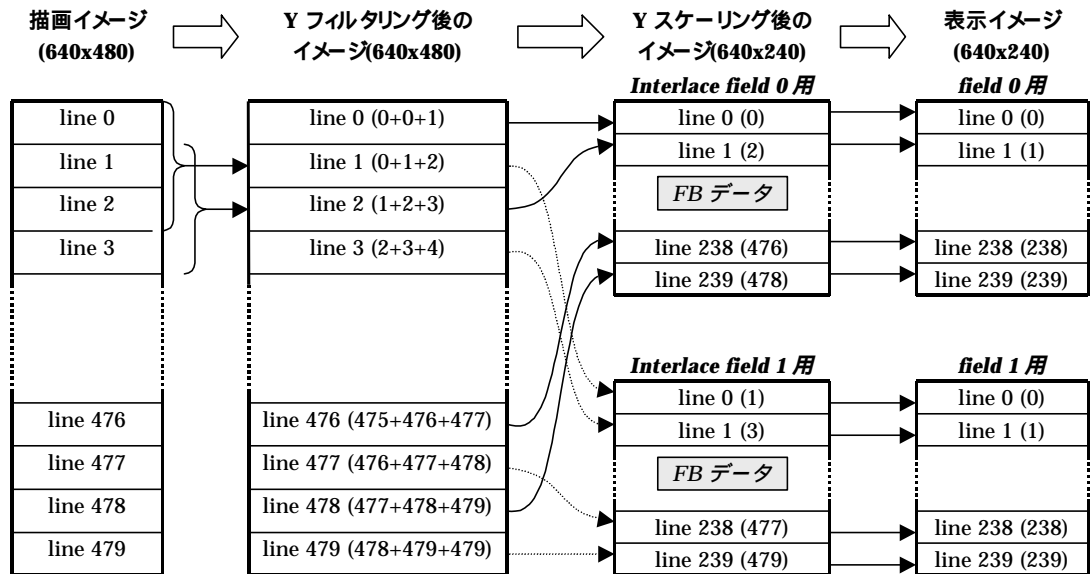


図 3-33 タイプB フリッカフリー

### § 3.22 カラー拡張とディザ処理

PowerVR2 では 16Bit カラーの拡張を 2 回行います。これは Accumulation Buffer での描画処理と TV への表示は 32Bit カラーで扱われるためです。

まず、1 回目の拡張は Accumulation Buffer での描画、エフェクト処理時に行われます。テクスチャはパレット形式 (32Bit でも扱える)、YUV 形式を除き RGB ダイレクトの 16Bit カラーで扱われますので、この描画処理時にカラー拡張が行われます。

この後、Frame Buffer にコピーされるのですが、この時に Frame Buffer モード設定により、32Bit カラーでそのままコピーも出来たり、16Bit カラーに減色してコピーすることも出来ます。

16Bit カラーでそのまま減色した場合は、マツハバンドとよばれる色階調ノイズが発生しますので、この場合にディザ処理を行うことで擬似的に 32Bit あるかのごとく表示することができます。

このディザ処理された画像は、表示される際にもう一度 32Bit カラーに復元されます。実際にデザインした色調と表示される画像の色調が若干異なるのはこのために起こります。

### § 3.23 カラークランプ

フレームバッファに描画する前に、ポリゴンリスト内にあるクランプカラーと比較して描画することが可能で、比較結果はその条件によってクランプカラーに固定されます。これにより、ポリゴン単位でフェードイン / アウトの効果を出すことが可能になります。

### § 3.24 Render Texture (レンダーテクスチャ)

本来 Accumulation Buffer の描画結果は Frame Buffer にコピーしますが、これを Texture Buffer にコピーすることでフレームバッファイメージをテクスチャに使用することが出来ます。ただし、このテクスチャをポリゴンに張る場合は、描画条件によっては 1 フレーム前の画面が表示されることになります。

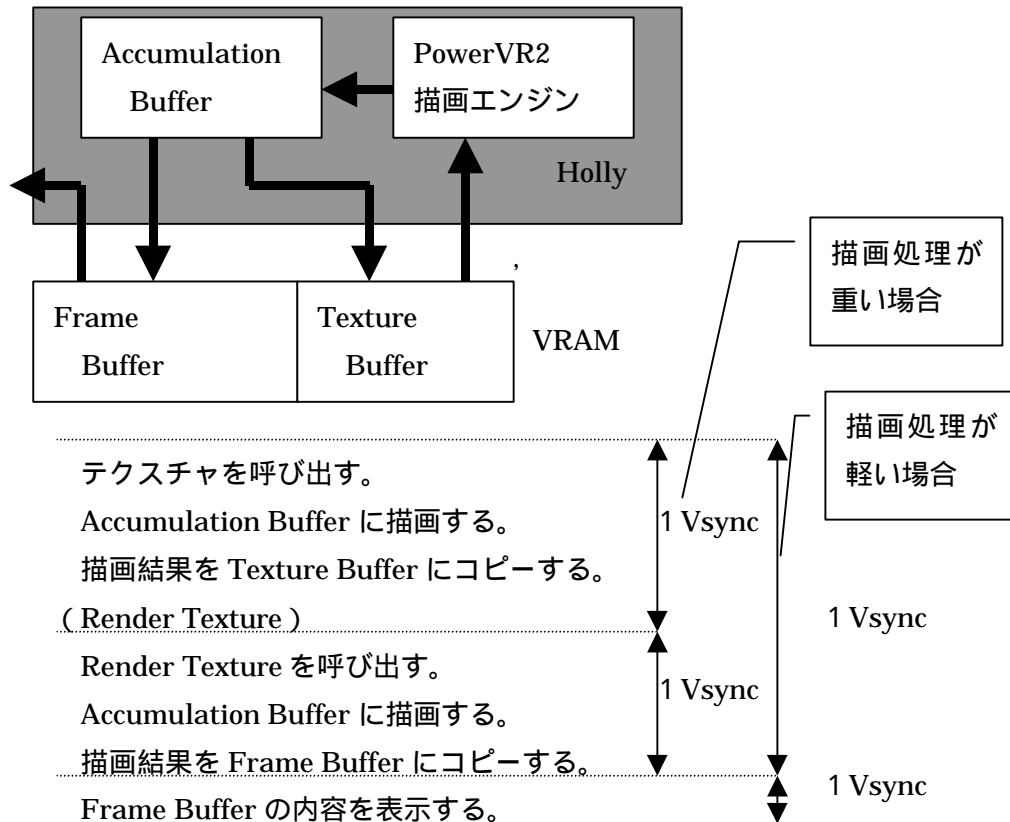


図 3-34 Render Texture の処理の流れ

Render Texture で定義されるテクスチャフォーマットは Non-Twiddled 形式となります。

### § 3.25 デジタルビデオエンコーダ

デジタルビデオエンコーダは、Holly チップから出力されたデジタル RGB ( 888 ) のデータを、すべてデジタル処理によってアナログ RGB , YC 信号 ( S 端子 ) , Video 信号に変換する IC です。この IC、は NTSC , PAL の放送規格に準じた映像信号を再現するため、ノイズの少ない非常に良好な絵を再現することが出来ます。

Dreamcast 用のエンコーダとしての特徴は、通常の放送規格に対し約 10% 明るさを強調してあります。これによって輝く様な白 ( 100% 時 ) を表現出来ますが、逆にそれによって TV , プロジェクタ TV 画面の焼きつげがおきますので、100% 白はここ一番のときの表現に利用する様にしてください



## § 4 サウンド機能

### § 4.1 サウンド機能の概要

- PCM 音源内蔵
- 128 ステップの DSP を搭載
- デジタルミキサー搭載
- 外部 Digital Audio 入力を 1 系統サポート
- PCM のデータフォーマットは 8,16bit Linear / 4bit ADPCM
- (ADPCM の方式は YAMAHA のオリジナル方式)
- スロット別の独立した LFO (Low Frequency Oscillator) を搭載
- 4セグメントの EG (Envelope Generator) を 64ch 搭載
- 4セグメントの EG によってカットオフ周波数を時変動する LPF を搭載
- フォワードループ機能搭載
- 最大 64 音同時発音
- 最大 1 オクターブまでピッチチェンジ可能な ADPCM

### § 4.2 ADPCM とは？

ADPCM の話をする前に音について簡単に説明し、ADPCM の理論的な背景となっている PCM の説明をします。その後 ADPCM の説明をします。

音というのは、空気中を伝わるいろいろな波長の波の合成であることが分かっています。  
この波形を絵にすると

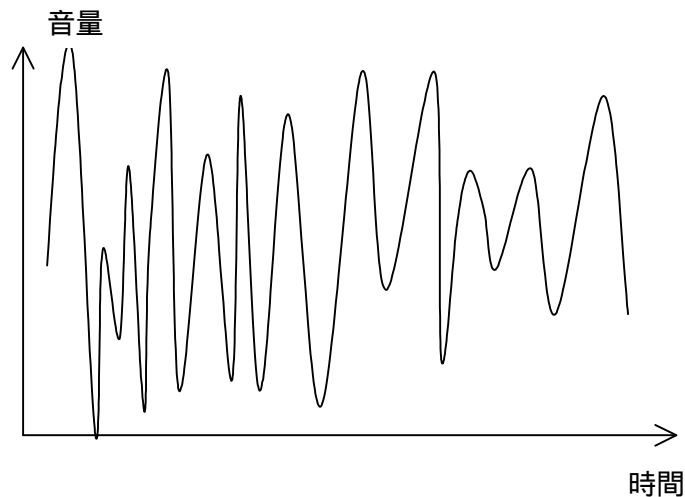


図 4-1 音の波形

というような、時間と音量の関係で表すことができます。

PCM とは、ある特定の時間でどれだけの音量だったかを記録する方法で、たとえば 44.1kHz の PCM では、1 秒間を 44100 等分して、 $1/44100$  秒のときの音量はいくつ、 $2/44100$  秒のときの音量はいくつといった具合に記録されます。

当然、1 秒の音には 44100 個の測定点が存在し、それらすべての測定点での音量が記録されます。

通常ここで測定された音量値はアナログな値で、たとえば 16db という感じです。これをデジタル化し、たとえば 0 から 255 までとか、0 から 65535 までというように範囲を決めてその範囲の値で記録します。

この例の 44.1kHz という数字をサンプリング周波数、音量のデジタル数値の範囲を量子化ビット数 (例えば 0 から 255 (=2 の 8 乗) までならば 8 ビット) といいます。

詳しくは下図 (図 4-2) をご覧ください

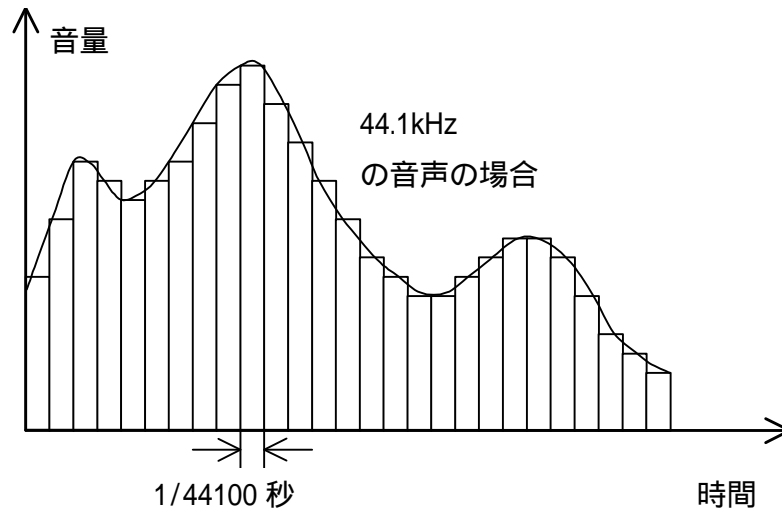


図 4-2 PCM のデータとは？

ADPCM 方式とは ある時間の音量を測定値で表すのではなく、前の結果から推測される音量値と実測値との差をその時間での音量の情報として記録する方法です。

測定した音量値よりも予測した音量値との差分のほうが変化する値の幅が小さいので、その分少ない記録量で PCM 方式と同じ波形を再現できます。

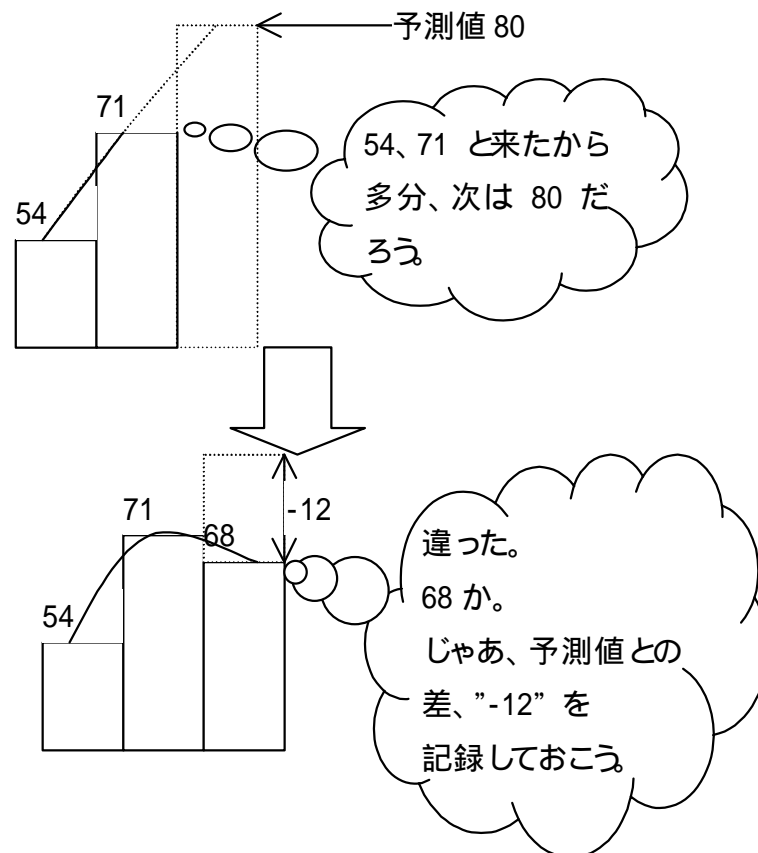


図 4-3 ADPCM とは？

上図 (図 4-3) を見て分かると思いますが、人によっては 54、71 という数字が来た後に、80 ではなく 78 という人もいれば 75 と予測する人もいるかも知れません。

そうです、この予測の方法は実際さまざまであり、例えば有名なものと Philips 社の予測方法は一般に XA 規格の一部として知られています。ただし圧縮したときと解凍したときで同じ予測の仕方をしなければ、元の音声の波形と圧縮解凍したときの波形が異なってしまいます。そのため ADPCM の圧縮ツールが手元にあるからといってそれを安易に使うことはできないのです。

Dreamcast の予測の仕方は Dreamcast で使用しているサウンドの LSI を開発した YAMAHA 社のオリジナルの予測方法を使用しています。

この YAMAHA 方式の予測方法を使った ADPCM の圧縮をするためのツールはセガから提供されます。

## § 5 ペリフェラル

Dreamcast には4つのペリフェラルの差込口があり、それぞれの差込口にはアナログコントローラや GUN などの入力装置を接続できるほか、専用コントローラには、液晶の小さな画面を備え外部バックアップメモリとしての機能を持つビジュアルメモリという装置を装着することもできます。

ペリフェラルポートは通称 Maple (メイプル)と呼ばれるセガオリジナルのシリアルプロトコルで接続され、最大約 2Mbpsの転送能力を持っています。

### § 5.1 ビジュアルメモリ

ビジュアルメモリは、小型携帯ゲーム機とバックアップメモリが一つになったもので、バックアップデータの保存や小型ゲーム機としての機能以外に、ビジュアルメモリ同士を接続することによって手軽にデータ交換が行えるような設計が施されています。

また、コントローラを接続した場合、それぞれのプレイヤーのコントローラについているビジュアルメモリにはそれぞれ別々のメッセージを表示したり、画面を表示できるので、複数プレイヤーが同時に遊んだときに個々のプレイヤーに違う情報を伝えることができます。



図 5-1 ビジュアルメモリ

### § 5.2 標準コントローラ

標準コントローラは、アナログ入力が標準でサポートされています。アナログ部分は、方向入力スティック、L Rキーの3系統がサポートされているため、ドライブゲーム等の操作性向上に一役かっています。デジタルキーとアナログキーは同時使用することが可能となっています。

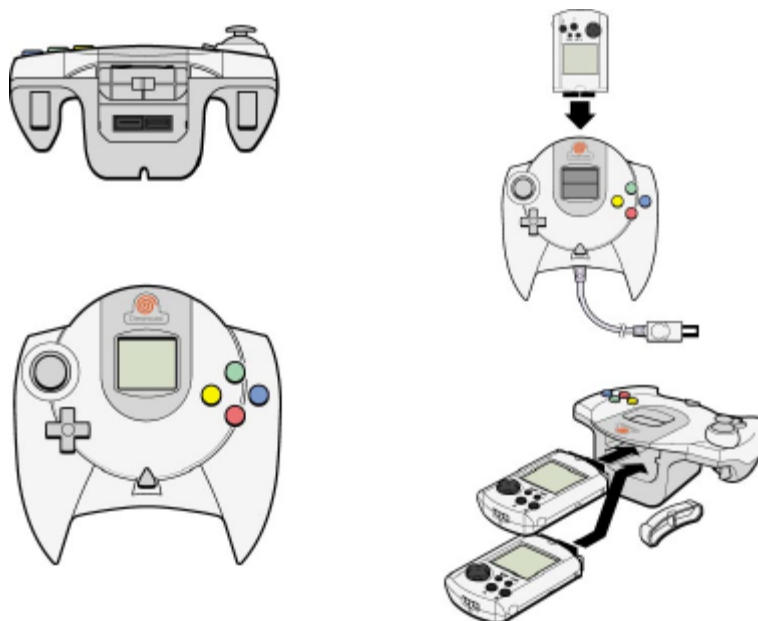


図 5-2 標準コントローラ

### § 5.3 対応予定のペロフェラル

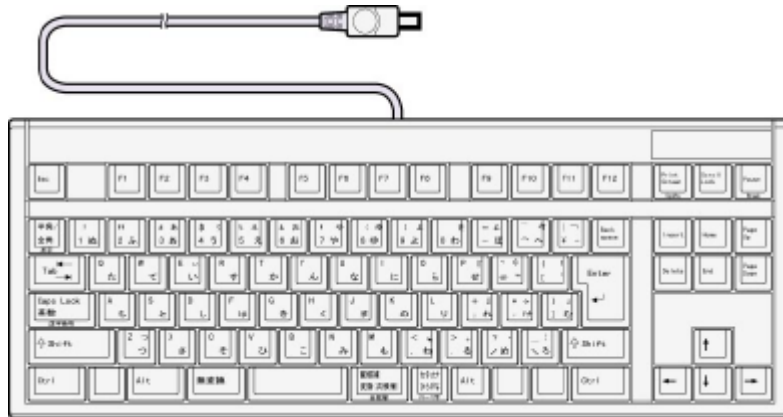


図 5-3 Key Board

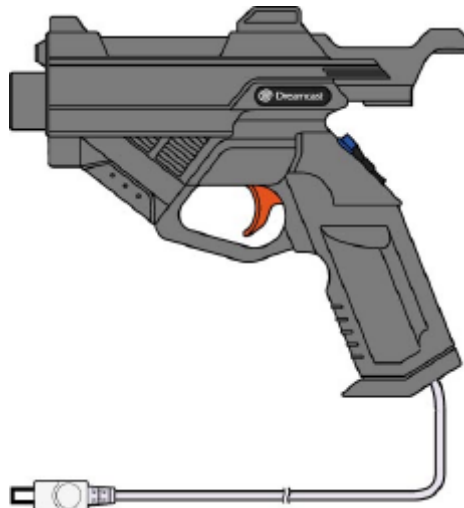


図 5-4 ガンコントローラ

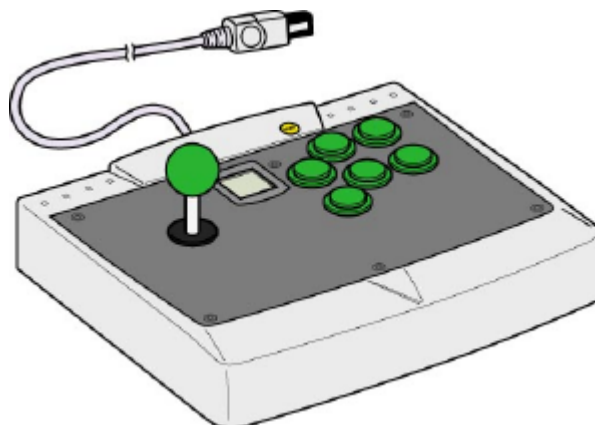


図 5-5 ジョイスティック

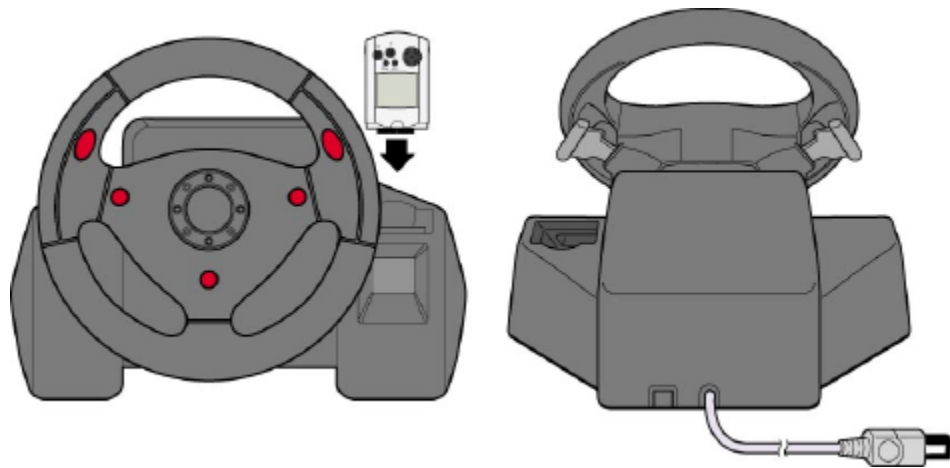


図 5-6 ハンドルコントローラ

## § 6 GD-ROM (仮称)

### § 6.1 GD-ROM とは？

Dreamcast の CDROM は GD-ROM (Giga byte Disk ROM) と呼ばれ、独自のフォーマットを採用しています。

回転速度が一定の機構で、同じ読み取り時間でも、データの読み取り部分 (レコードの針に相当する部分) が外側ほど、大量のデータを読み取ることができます。(通常の CD-ROM ドライブは線速度一定という形式を使用しているので、読み取り部分がどこにあっても同じ時間内に読みとれるデータの量は一定です。その代わり、読み取り部分の位置によって回転速度が変化します。ディスクドライブの部分に耳を当てれば、ディスクの回転音が違うのがわかるでしょう)つまり、一度に大量のデータを読み取る必要があるときは、そのデータを GD-ROM の外側に置くほうが高速に読み取れます。

このような理由で、Dreamcast で採用している CD-ROM は、4 ~ 12 倍速と速度が変化します。最近の CD-ROM ドライブでは、"12 ~ 24 倍速" といった表記のものがありますがこれらは同じ方式を採用した CD-ROM ドライブです。

### § 6.2 GD-ROM の構造

ディスクは構造上 3 つのブロックに分かれています。

一番内側を単密エリアと呼び、ここには一般的な CD-ROM の規格に従った形式でデータや CDDA が収められます。

当然、CD プレイヤーやコンピュータでこのエリアを参照することが可能です。

このエリアは 4 分間程度しかないので、例えば

「このディスクはセガ Dreamcast 用のゲームディスクです。ゲームのデータが入っていますので再生しないでください。」

といった、SATURN でも行っていたようなメッセージを入れる場合が想定されています。

また、Dreamcast 用のソフトからこの領域にアクセスすることはできません。

次のエリアは高密エリアで

ここにゲームのデータやプログラム (ゲームで使用する) CD-DA のデータなどを収納することができます。

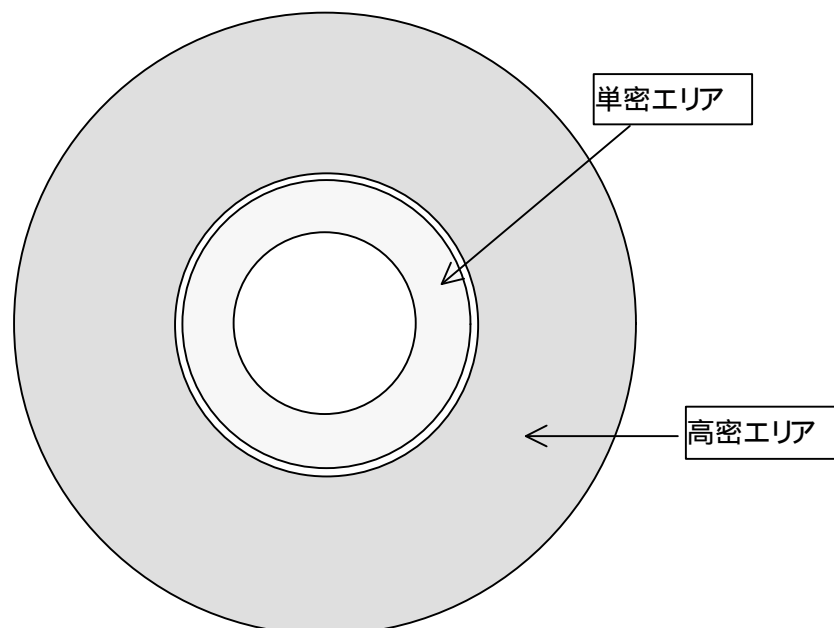


図 6-1 GD-ROM の構造

## § 7 CPU

Dreamcast には HITACHI 製の SH7091 という CPU が搭載されます。SH7091 は外部クロック 100MHz 内部クロック 200MHz で動作し

**FPU (浮動小数点計算モジュール)**

**MMU (仮想メモリ管理モジュール)**

**DMA コントローラ**

**タイマ**

**命令 8KB データ 16KB のキャッシュ**

**5 段のパイプラインと 2 本のスーパースカラ**

などの機能を持っています。

### § 7.1 FPU

FPU は、小数点を含む数値の加減乗除を高速に処理するためのモジュールで、3D の座標計算など通常は時間がかかる数値処理を専門に行います。内部的には、32Bit の浮動小数点演算機を 4 つ搭載しており、3 次元画像処理に威力を発揮します。これらは、4 次元のベクトル変換を 7 サイクル、4 次元の内積演算を 5 サイクルで実行します。また、拡張機能として、セガプライベートファンクション(セガ専用命令セット)をサポートしております。

### § 7.2 MMU

MMU は実メモリの番地に対して仮想的な番地を割り当て、あたかも広大なメモリがあるように振る舞うことができます。

DragonOS はこの MMU モジュールを使用し、実メモリ空間をある程度隠蔽し、本来は分断されているメモリを仮想的に連続したメモリのように見せることができます。

使う側では、実際にはメモリが分断されているということを気にする必要はありません。

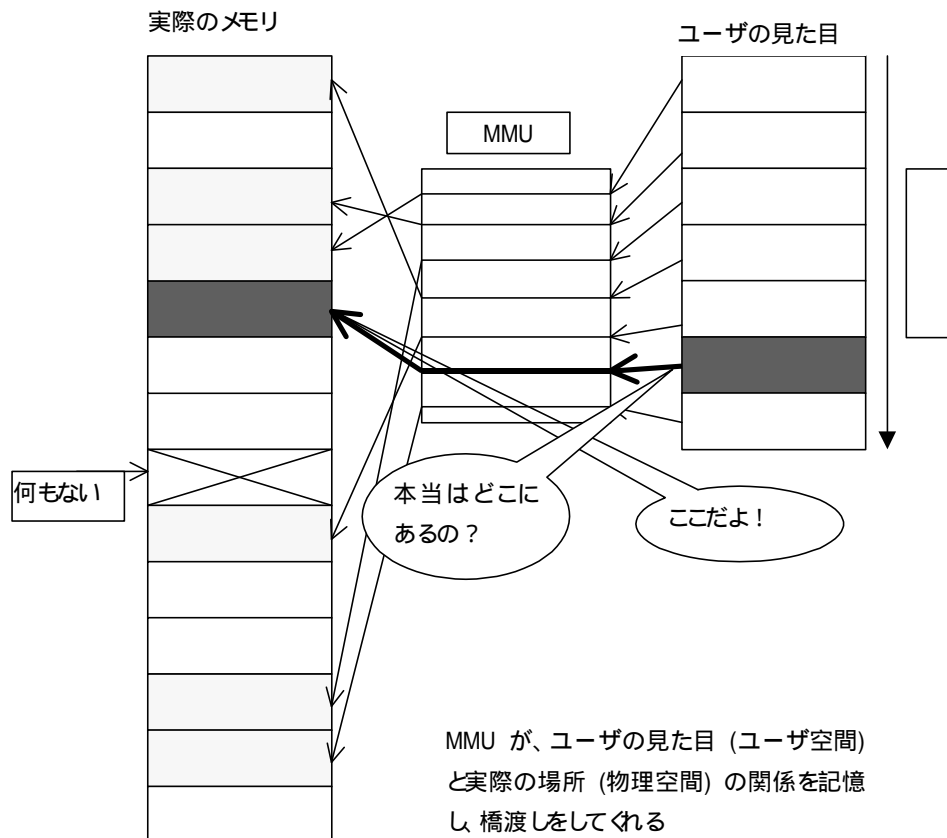


図 7-1 MMU の働き



MMU 用のアドレス変換バッファ (TLB) として

**命令 (データ用としても利用できる) : 4 エントリ**

**データ: 64 エントリ**

を備えています。

また、SH7091 は他の MMU を積んだ CPU と異なり、ページサイズ (MMU が扱える最小のサイズ) が可変で、

**1k、4k、64k、1M**

単位でこれらのページサイズの混在を認めています。たとえば

**0x00000000 ~ 0x0FFFFFFF はページサイズ 1K**

**0x10000000 ~ 0x1FFFFFFF はページサイズ 4k**

**0x20000000 ~ 0x7FFFFFFF はページサイズ 64k**

**0x80000000 ~ 0xFFFFFFFF はページサイズ 1M**

とすることができます。

単純にすべてのメモリ空間を 1k バイトのページサイズとした場合、Dreamcast のすべてのメインメモリをカバーするには

**16x1024=16384**

の変換テーブルが必要になります。しかし、SH7091 には高々 68 個の変換テーブルしかありません。内部で持っているテーブルに自分の探したいアドレス変換情報が存在しないとき、MMU はメインメモリ上に確保してあるマスターの変換テーブルを探し、内部のバッファに取り込みます。

しかし、この取り込み作業は内部バッファを参照するのに比べて非常に多くの時間がかかります。この例の場合、頻繁にマスターテーブルを見に行くため、処理の時間としては多くの無駄が発生します。

では、ページサイズを思い切って 1M にするとどうなるでしょう？

ページサイズが 1M のとき、必要なテーブルのエントリは

**16 個**

十分に内部バッファに入る大きさです。

ところがこの場合別の問題が発生します。

MMU は、メモリのアロケートが指示されたとき、ページサイズでメモリを確保します。

つまり、たとえば 16 バイトのメモリを確保するために、1M バイトの実メモリを確保するのです。

それはそれで非常に無駄が多いことがお分かりになるでしょう

つまり、大量のデータを必要とするところ、例えばムービーのバッファ領域などは大きいページサイズで、プログラムの領域やテンポラリのデータ領域 (ヒープ領域) などは小さ目のページサイズにすることで、MMU の内部変換テーブルを有効に利用し、かつメモリも上手に使用することが可能なのです。

しかしながら、通常これらの仕事は、OS などの管理を担当するモジュールが行うため、それほど意識する必要はありません。

## § 7.3 DMA

DMA は大量のデータのある場所から別の場所に一気に転送するもので、DMA がデータを転送している間 CPU の計算機能などは平行して動くことができるので、転送が終了するまで処理が中断することがありません。そのため時間を無駄なくしようすることが可能です。

SH7091 には DMA が 4 つありますが、それぞれのチャンネルは目的ごとに異なります。

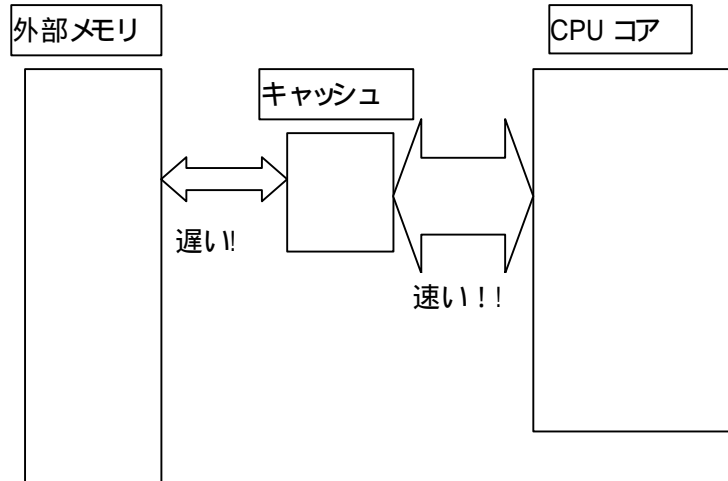
4 つのうち 2 つは、システムが使用するため、ユーザレベルで使用できるのは 2 つ分です。システムで予約している DMA のうちひとつは、高速にテクスチャメモリにアクセスするための DMA で、メインメモリから、テクスチャ RAM への一方向の転送にしか使用できません。システムで予約しているもうひとつの DMA は、各外部デバイスとメモリとの間で転送を行うもので、これにはテクスチャ RAM からメインメモリへの転送も含まれます。

## § 7.4 タイマ

SH7091 は内部に独立して動くタイマを内蔵しています。

## § 7.5 キャッシュ

SH7091 は命令キャッシュ 8KB、データキャッシュ 16KB を持ち、時間がかかる外部メモリへのアクセスを減らすことができます。ただし、SH7091 のキャッシュは Pentium , PowerPC などの CPU とは異なり、ダイレクトマップ方式を採用していますので、キャッシュのヒット率が良くありません。



キャッシュは、外部メモリの内容をアドレス (場所) の情報と一緒に保存している。(沢山ほどよい。)

最初は (時間がかかる) 外部メモリにアクセスしても、次に同じところの情報を引き出すときは、キャッシュに溜まっている内容を読みに行くので速い。

図 7-2 キャッシュの利点

Dreamcast に使用しているメモリはシンクロナス DRAM と呼ばれるもので、高速アクセスするためにクロック (100MHz) に同期したバースト転送 (データを指定した先頭アドレスから連続で R/W を行う) を行います。この転送サイズは1回あたり 32Byte で、キャッシュにミスヒットした場合は、たとえ必要なデータサイズが4Byte であっても、32Byte 分のアクセスが入ります。このため、約 10 倍も余計に時間がかかります。

**キャッシュを意識したプログラミングをすることが重要になります。**

## § 7.6 ストアキュー

ストアキューとは、外部メモリに対し高速転送を行うための Write 専用のバッファです。このバッファは

**32Byte × 2 バッファ**

のダブルバッファ構成となっており、片方のストアキューがメインメモリに対し転送を行っている間、もう一方のストアキューに対し CPU からの Write を行えます。

## § 7.7 パイプライン・スーパースカラ

SH7091 にはスーパースカラと呼ばれる機能が備わっています。

これを簡単に説明すると、昔の CPU は、命令 (受注) を受けてから、その処理が完了 (納品) するまで、別の命令を受け付けることはできませんでした。それは、すべての工程が 1 つの機械によって処理されていたからと考えるとよいでしょう。

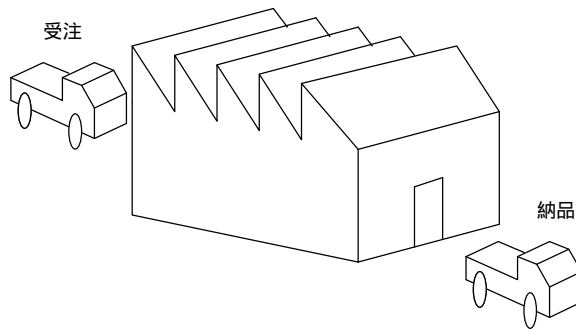
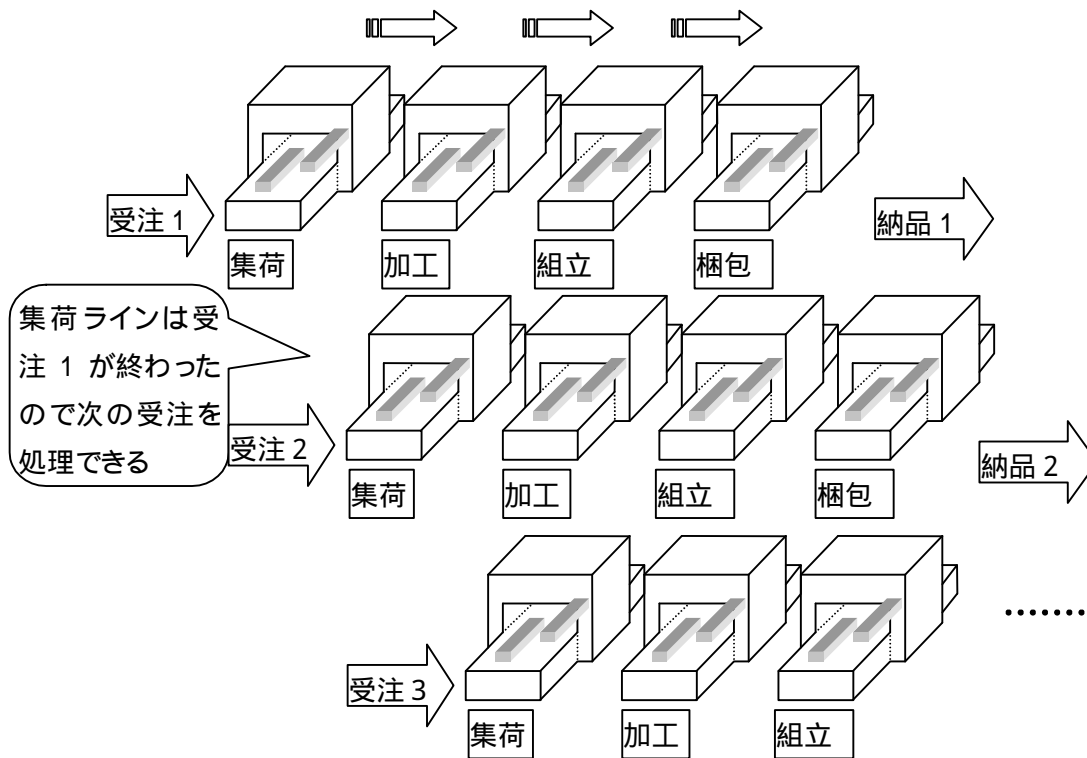


図 7-3 初期の CPU

さて、それでは効率が悪いので、作業工程を見直して、いくつかの独立した工程に分けました。  
図 30 に従えば、各工程を専門で行う機械を導入し、機械ごとに仕事を振り分けたと考えてください。  
これで、見かけ上複数の受注を捌いているように見えます。



たとえば、集荷のラインは、自分の作業が終われば暇になるので、次の受注を処理できる。加工や組立、梱包のラインもそれぞれ独立に動ける。

図 7-4 パイプライン処理 (図では 3 本のパイプライン)

SH7091 の 5 本のパイプラインとは、このように 5 つの命令を同時に処理する能力のことを指します。  
ただし、気を付けなければいけないのは、前の注文で作った品物を使用したものを次の注文の材料として使いたい場合で、この場合は、次の注文の材料は前の注文が終わるまでは、取り掛かることはできないので、作業効率が落ちます。(つまりそういう作り方を極力避けるのがよいパフォーマンスを出すためのよい方法なのです。)

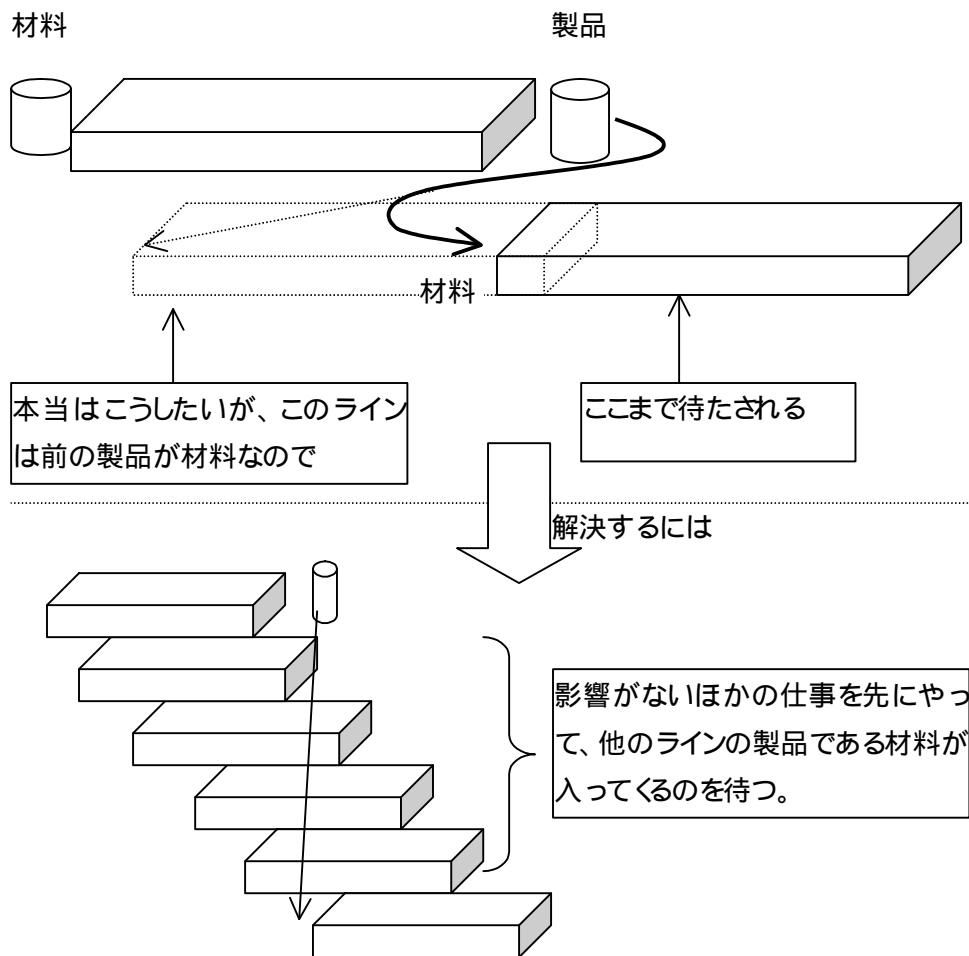


図 7-5 効率のよいパイプラインの使い方

さて、さらに効率を上げるにはどうしたらいいでしょう？  
一番簡単な方法としては、工場をもう1つ建設してしまうことです。  
SH7091 のスーパースカラとは、この工場を増やすという方法に当たるもので、2つの命令を全く同時に捌くことができるのです。

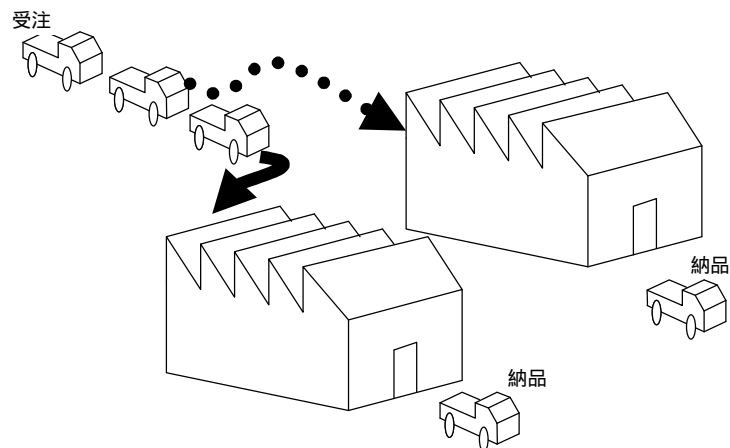


図 7-6 スーパースカラ

当然各工場は、工程が分離（パイプライン化）しており来た受注を効率よく捌くことができるようになってい

ます。  
ただし、この場合も先に挙げたような前の注文で作った商品を後で使うという場合には作業が止まってしま

います。  
このことは CPU を効率よく使うためのノウハウなのです。

もうひとつの SH7091 の高速化テクニックはだらだらやっていた仕事をてきぱきと終わらせる（クロックアッ  
プ）ことです。SH7091 のクロックは 200MHz、SATURN で使用していた SH2 が 28MHz で動いていたこと  
を考えると、実に 7 倍もてきぱき動いていることになります。

## **§ 8 MODEM**

Dreamcast は、家庭用ゲーム機としては初めて標準で Modem を搭載しています。

< 主な仕様 >

- ・ Full Duplex V.34 (33.6Kbps ) Data Modem
- ・ エラー訂正 データ圧縮は MNP2 5 , V42 , V42 bisをサポート
- ・ エラー訂正 データ圧縮はソフトウェア処理
- ・ コントロール用のマイコンを含まない Passive Modem

このモデムは リムーバブル方式(取り外し可能)になっており 将来的には更に高速な Modem への交換が可能な設計になっています。