Wii Graphics Primer

Version 1.0.0

RVL-06-0301-001-A
Released: February 15, 2008

Contents

1 Introduction 9
1.1 Overview 9
1.2 Document Structure 9
2 Special Features of Wii Graphics 12
2.1 Introduction to Wii Graphics Concepts 12
2.2 Main Features of Wii Graphics 12
2.3 Unsupported Graphics Features 13
3 Graphics Creation Process Flow 14
3.1 Function of the Graphics Processor Unit (GPU) 14
3.2 Vertex Data and Coordinate Conversion 15
3.2.1 Vertex Data 15
3.2.2 Conversion of Vertex Position Coordinates 16
3.2.3 Primitives Types 17
3.2.4 Front Surface of Polygon 17
3.3 Shading 19
3.3.1 Rasterized Color 19
3.3.2 Lights 20
3.4 TEV (Texture Environment Unit) 22
3.4.1 What is the TEV? 22
3.4.2 Registers that Can be Used with TEV Stage Computations 24
3.4.3 TEV Stage Computations 26
3.4.4 Swap Table 30
3.4.5 Fog 32
3.5 PE (Pixel Engine) 33
3.5.1 What is the PE? 33
3.5.2 Z Comparison 33
3.5.3 Blending 34
3.5.4 Dithering 35
3.6 Video Output 36
3.6.1 Frame Buffer 36
3.6.2 Antialiasing 39
3.6.3 Gamma Correction 40
3.6.4 Deflicker 40
3.6.5 Converting From RGB to YUV 40
4 Examples of TEV Settings 41
4.1 Modulation 41
4.1.1 Features of Modulation 41

4.1.2 What is Modulation? 42
4.2 Blending Using Two Textures 43
4.2.1 Addition 44
4.2.2 Subtraction 46
4.2.3 Multiplication 48
4.2.4 Addition and Multiplication Combined 50
4.2.5 Decal 52
4.2.6 Decal (Applied) 54
4.2.7 Proportional Blending 56
4.3 Two-Color Interpolation 58
4.3.1 How Two-Color Interpolation Works 58
4.3.2 Stage Settings for Two-Color Interpolation 59
5 Textures 60
5.1 Texture Size 60
5.2 Texture Formats 61
5.2.1 Opaque, Translucent, and Outline 61
5.2.2 Types of Texture Formats 62
5.2.3 Optimal Texture Format 64
5.2.4 Intensity 65
5.2.5 Intensity Alpha 66
5.2.6 RGB 66
5.2.7 RGBA 67
5.2.8 Compressed Texture 68
5.2.9 Color Index Texture 71
5.2.10 Tiles 74
5.3 Repeat Pasting of a Texture 75
5.3.1 Clamp, Repeat, and Mirror 75
5.3.2 Restrictions for Specifying Repeat and Mirror 75
5.4 Mipmap 76
5.4.1 Mipmaps 76
5.4.2 LOD Bias 77
5.5 Texture Filters 78
5.6 Texture Mapping 80
5.6.1 Texture Coordinate Space and Texture Coordinates 80
5.6.2 Converting and Mapping Texture Coordinates 81
5.7 Mapping Multiple-Layered Textures (Multi-Texturing) 85
5.8 Indirect Textures 86
5.8.1 What is an Indirect Texture? 86
6 Overview of Wii Graphics Hardware 88
6.1 Hardware Configuration 88

6.1.1 Broadway 88
6.1.2 Hollywood 89
6.1.3 External Main Memory (MEM2) 89
6.1.4 Graphics Memory 89
6.2 Performance 89
6.2.1 Processes that Have the Most Impact on Performance 89
6.2.2 Processes that Do Not Impact Performance 89

Figures

Figure 1-1 Screen Display, Example 1 10
Figure 1-2 Screen Display, Example 2 11
Figure 3-1 GPU Process Flow 14
Figure 3-2 Data that Can be Held by One Vertex 15
Figure 3-3 Coordinate Conversion of Vertex Position Coordinates (For Programmers) 16
Figure 3-4 Primitive Types 17
Figure 3-5 Polygon Surfaces 17
Figure 3-6 Association of Lights and Color Channels 19
Figure 3-7 Types of Light 20
Figure 3-8 Formulas for Calculating Light (For Programmers) 21
Figure 3-9 TEV Block Diagram 22
Figure 3-10 Example of Output when Combining Texture with Rasterized Color 23
Figure 3-11 Positioning Color Registers 24
Figure 3-12 Features and Uses of Color Registers 24
Figure 3-13 Positioning of Constant Registers 25
Figure 3-14 Features and Uses of Constant Registers 25
Figure 3-15 Color and Alpha Stages 26
Figure 3-16 TEV Computation Expression 26
Figure 3-17 Compare Mode 27
Figure 3-18 Color Stage and Input Sources 27
Figure 3-19 Example of Stage Settings for Applying Shadows to Textures 28
Figure 3-20 Alpha Stage and Input Sources 28
Figure 3-21 Output of the Stage Computation Result 29
Figure 3-22 The Swap Feature 31
Figure 3-23 Fog Function Graph 32
Figure 3-24 Correction of Fog Range 32
Figure 3-25 Examples of Z Comparison 34
Figure 3-26 Expression for Blending 34
Figure 3-27 Dithering 35
Figure 3-28 Scaling Frame Buffer 36
Figure 3-29 Interlace Mode 37

Figure 3-30 Interlace (Field Rendering) Mode 38
Figure 3-31 Noninterlace Mode 38
Figure 3-32 Progressive Mode 39
Figure 3-33 Antialias 39
Figure 3-34 Gamma Correction 40
Figure 4-1 Modulated Images 41
Figure 4-2 Blending of Two Textures 43
Figure 4-3 Example of Texture Addition 44
Figure 4-4 Example of Stage Settings for Addition 45
Figure 4-5 Example of Subtraction 46
Figure 4-6 Example of Stage Settings for Subtraction 47
Figure 4-7 Example of Texture Multiplication 48
Figure 4-8 Example of Stage Settings for Multiplication 49
Figure 4-9 Example of Addition and Multiplication Combined 50
Figure 4-10 Example of Stage Settings for Addition and Multiplication Combined 51
Figure 4-11 Example of Decal 52
Figure 4-12 Example of Stage Settings for Decal 53
Figure 4-13 Example of Decal (Advanced) 54
Figure 4-14 Example of Stage Settings for Decal (Applied) 55
Figure 4-15 Example of Proportional Blending 56
Figure 4-16 Example of Stage Settings for Proportional Blending 57
Figure 4-17 Example of Two-Color Interpolation 58
Figure 4-18 Example of Stage Settings for Two-Color Interpolation 59
Figure 5-1 Texel 60
Figure 5-2 Minimum and Maximum Sizes of Textures 60
Figure 5-3 Opaque, Outline, and Translucent Textures 61
Figure 5-4 Comparison of Texture Format and Texture Size (256x256 Texture) 63
Figure 5-5 Optimal Texture Formats 64
Figure 5-6 Graphic Suited For the Compressed Texture Format 68
Figure 5-7 Graphic Not Suited For Compressed Texture Format 69
Figure 5-8 Compressed Texture (Generating Two Interpolation Colors) 69
Figure 5-9 Compressed Texture (Using Outline) 70
Figure 5-10 Color Index 71
Figure 5-11 Switching Color Palettes 71
Figure 5-12 Texture Format and Tiles 74
Figure 5-13 Tiling Method 75
Figure 5-14 Mipmapping Used to Erase Moiré Patterns 76
Figure 5-15 Illustration of Mipmap Texture 76
Figure 5-16 Using LOD Bias to Adjust the Mipmap Level Boundaries 77
Figure 5-17 Point Sampling and Bilinear Filtering 78
Figure 5-18 Texture Coordinate Space 80
Figure 5-19 Using Texture Coordinates for Mapping 80

Figure 5-20 Converting Texture Coordinates....81
Figure 5-21 Using Texture Matrices to Rotate and Scale a Texture....81
Figure 5-22 Illustration of Environment Mapping....82
Figure 5-23 Example of Projection Mapping....82
Figure 5-24 Toon Shading....83
Figure 5-25 Toon Shading, Mechanism 1....83
Figure 5-26 Toon Shading, Mechanism 2....84
Figure 5-27 Mapping of Multiple, Layered Textures....85
Figure 5-28 Mapping with Different UV for Each Texture....86
Figure 5-29 Example of Displaying with Indirect Texture....86
Figure 6-1 Block Diagram of Wii's Graphics Hardware....88

Tables

Table 3-1 Data that Can be Held by One Vertex....15
Table 3-2 Types of Fog Function....32
Table 3-3 Types of Z Comparison Expressions....33
Table 3-4 Examples of Blending Coefficients....34
Table 3-5 Pixel Formats of the EFB....36
Table 5-1 Comparison of Texture Formats....62
Table 5-2 Intensity Format Textures....65
Table 5-3 Intensity Alpha Format Textures....66
Table 5-4 RGB Format Textures....66
Table 5-5 RGBA Format Textures....67
Table 5-6 Compressed Texture Format Textures....68
Table 5-7 Color Index Format Textures....73
Table 5-8 Texture Filter Modes....79

Revision History

Version	Revision Date	Description
1.0.0	2007/11/16	Initial version.

1 Introduction

1.1 Overview

This document introduces the main features of Wii graphics to developers and designers of Wii software.

To make information readily accessible to more readers, we have omitted the descriptions of some features. For a more detailed explanation, see the Graphics Library and other documentation included in the Revolution SDK.

This document also includes supplementary information for NintendoWare for Revolution (henceforth referred to as NintendoWare), which is the integrated development environment for Wii. Information specific to NintendoWare is enclosed in a frame and labeled with the NW4R logo.

1.2 Document Structure

This document is divided into four topics.

- Special Features of Wii Graphics
- Graphics Creation Process Flow
- Examples of TEV Settings
- Textures

In the first half of the document, the basic mechanism for displaying Wii 3D graphics is explained. The second half of the document provides specific examples of material settings and information on textures.

The chapters in this document can be read in any order.

Figure 1-1 shows Wii software reference images. This document does not discuss the settings and data used in games.

Figure 1-1 Screen Display, Example 1

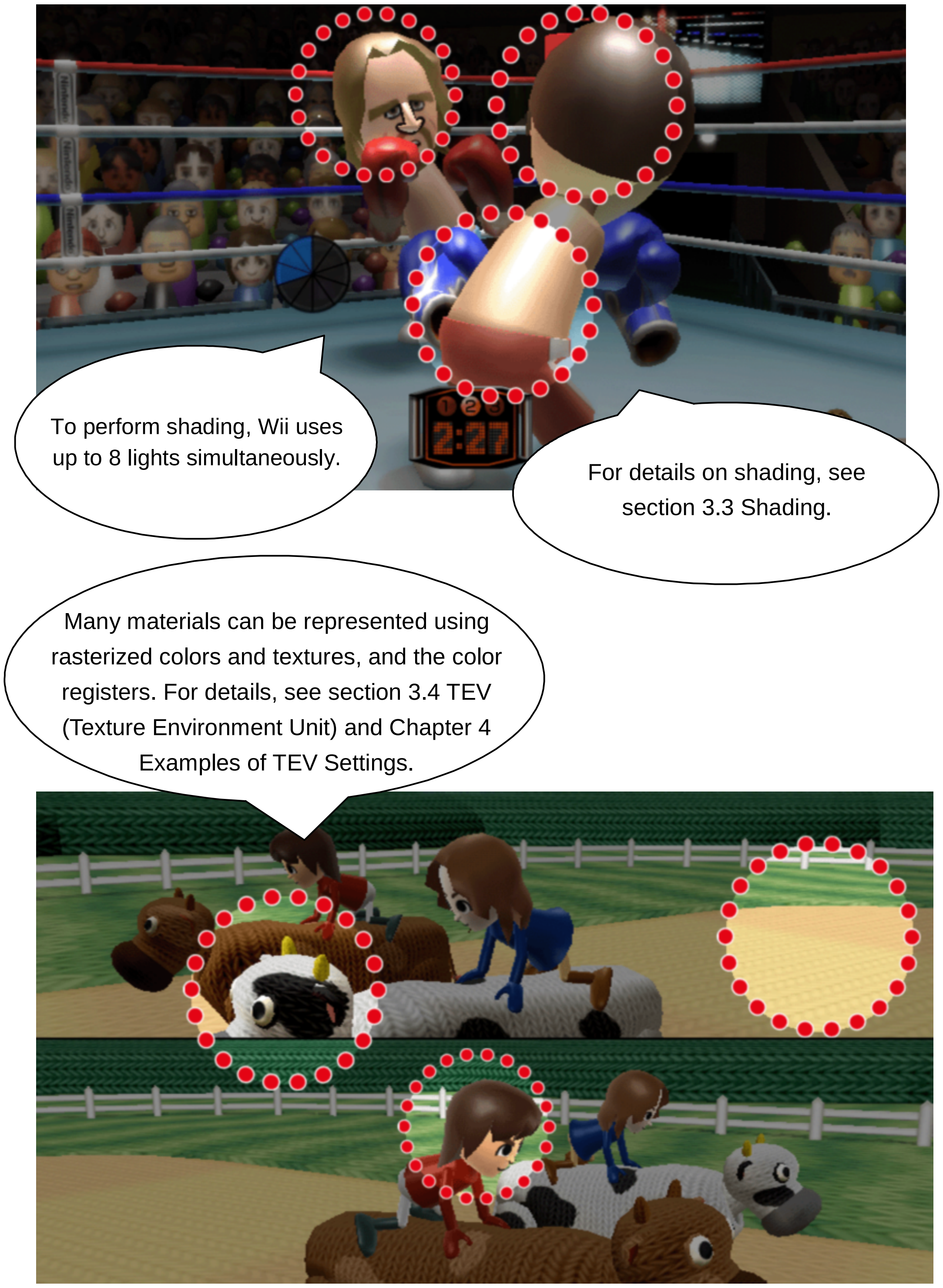
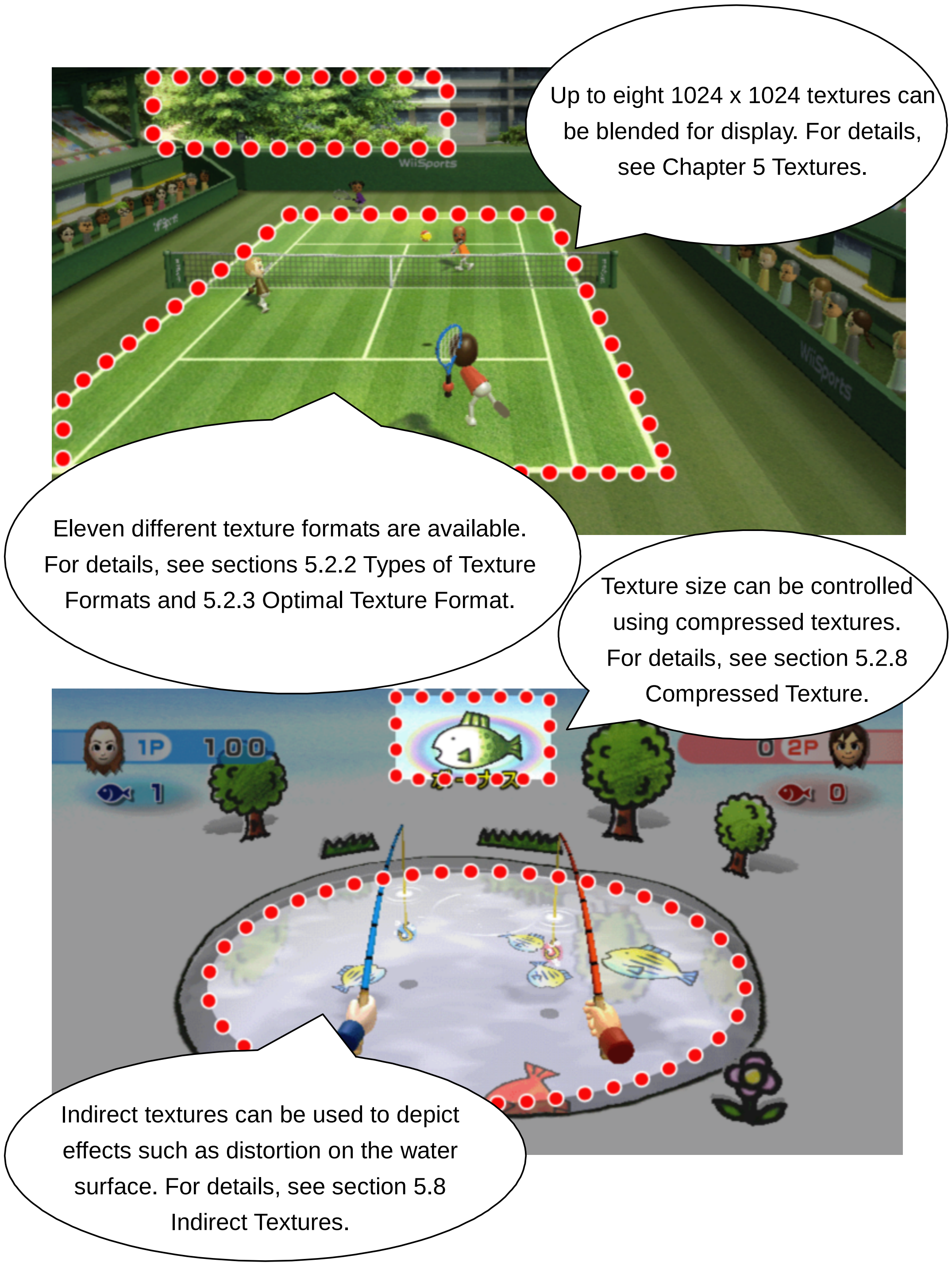

To perform shading, Wii uses up to 8 lights simultaneously.
For details on shading, see section 3.3 Shading.
Many materials can be represented using rasterized colors and textures, and the color registers. For details, see section 3.4 TEV (Texture Environment Unit) and Chapter 4 Examples of TEV Settings.

Figure 1-2 shows Wii software reference images. This document does not discuss the settings and data used in games.

Figure 1-2 Screen Display, Example 2


Up to eight 1024 x 1024 textures can be blended for display. For details, see Chapter 5 Textures.
Eleven different texture formats are available. For details, see sections 5.2.2 Types of Texture Formats and 5.2.3 Optimal Texture Format.
Texture size can be controlled using compressed textures. For details, see section 5.2.8 Compressed Texture.
Indirect textures can be used to depict effects such as distortion on the water surface. For details, see section 5.8 Indirect Textures.

2 Special Features of Wii Graphics

This chapter summarizes the main features of Wii 3D graphics. More detailed information is provided in subsequent chapters.

2.1 Introduction to Wii Graphics Concepts

Wii inherits the graphics features of Nintendo GameCube™ and is designed to deliver stable performance at all times. The processing burden increases with the number of lights or TEV stages used simultaneously. Conversely, processing decreases as this number decreases.

2.2 Main Features of Wii Graphics

Wii has a 3D graphics engine for rendering polygons that includes the following features.

Vertex Data (see section 3.2.1 Vertex Data)

- Supports user defined vertex data format
- Supports changing the matrix used for each set of vertex position coordinates
- Supports setting up to eight UV coordinates for each vertex

Shading (see sections 3.3.2 Lights and 3.4 TEV (Texture Environment Unit)

- Supports use of up to eight lights simultaneously
- Supports use of light to express a point light source, parallel light source, or spotlight
- Supports light calculations for vertex color
- Supports expression of a variety of material effects using TEV formulas
- Supports use of four color registers and four constant registers
- Supports fog processing

Pixel Processing (see section 3.6 Video Output)

- Supports frame buffer sizes up to 640 (width) x 480 (height) (NTSC format)
- Supports antialiasing, deflicker, and gamma correction to the entire screen
- Can freely set alpha-compare, Z buffer, and blending processes

Textures (see section 5 Textures)

- Can select from 11 texture formats
- Supports high-quality compressed texture formats
- Supports use of textures with resolutions of up to 1024 (width) x 1024 (height)
- Supports layering of up to eight textures for display
- Supports selection from three methods of tiling (clamp, repeat, and mirror)
- Supports use of filter features such as point sampling and bilinear filtering
- Supports use of mipmap features with up to 11 levels
- Supports use of swap features to control texture's RGBA components
- Supports use of indirect features to distort textures for display
- In addition to UV mapping, supports other mapping with textures, including environmental and projection mapping

2.3 Unsupported Graphics Features

Wii does not support the following features and graphic representations.

Shading

- Programmable shaders
  - Vertex, pixel, and programmable shaders are not supported. TEV stage computation formulas must be used, instead
- Normal mapping

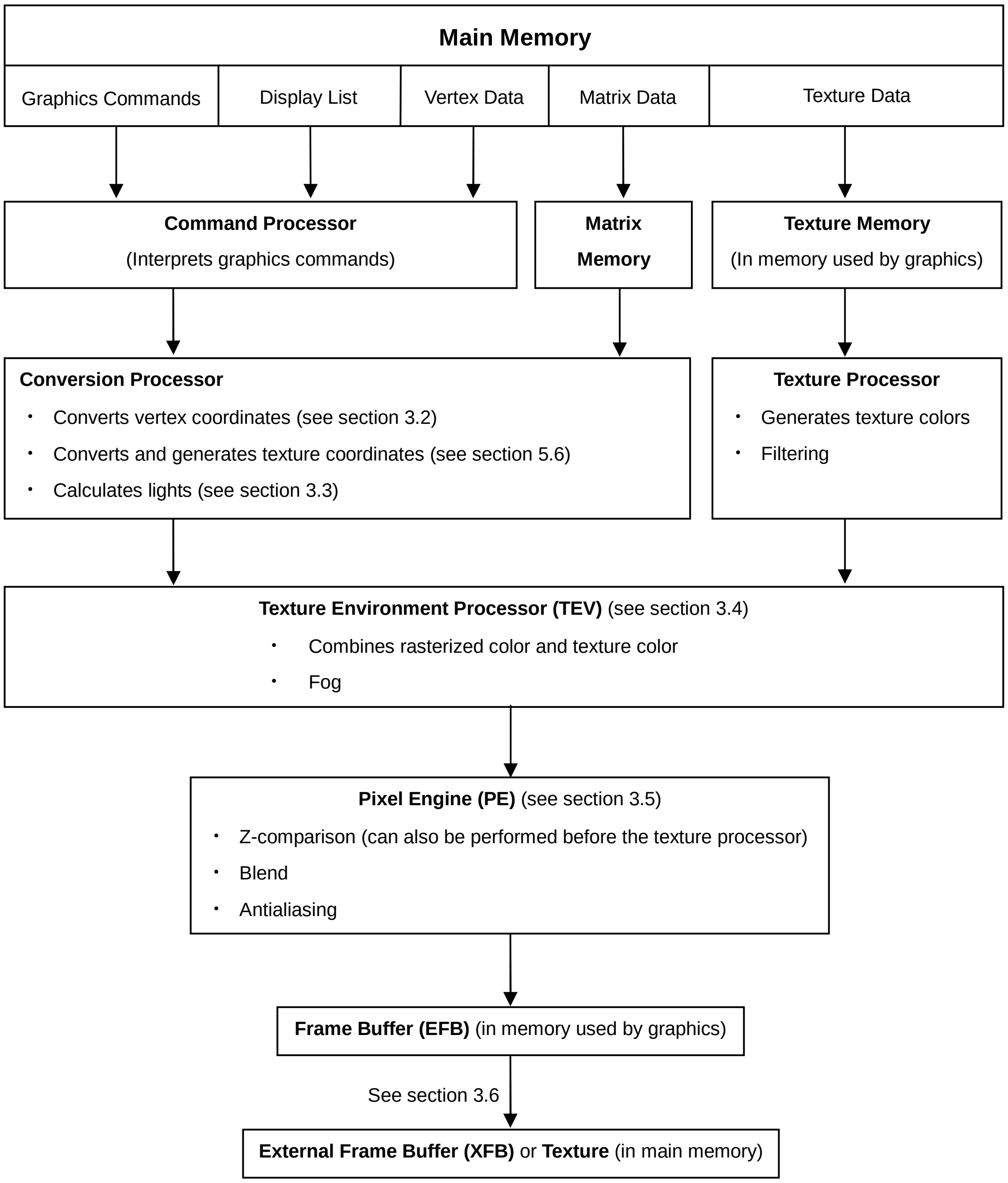
Indirect textures can be used to express normal mapping in object space with NintendoWare. But NintendoWare does not support normal mapping to envelope models.

3 Graphics Creation Process Flow

3.1 Function of the Graphics Processor Unit (GPU)

Figure 3-1 illustrates the process flow in the GPU.

Figure 3-1 GPU Process Flow


Main Memory
Graphics Commands
Display List
Vertex Data
Matrix Data
Texture Data
Command Processor
(Interprets graphics commands)
Matrix Memory
Texture Memory
(In memory used by graphics)
Conversion Processor
・ Converts vertex coordinates (see section 3.2)
・ Converts and generates texture coordinates (see section 5.6)
・ Calculates lights (see section 3.3)
Texture Processor
・ Generates texture colors
・ Filtering
Texture Environment Processor (TEV) (see section 3.4)
・ Combines rasterized color and texture color
・ Fog
Pixel Engine (PE) (see section 3.5)
・ Z-comparison (can also be performed before the texture processor)
・ Blend
・ Antialiasing
Frame Buffer (EFB) (in memory used by graphics)
See section 3.6
External Frame Buffer (XFB) or Texture (in main memory)

3.2 Vertex Data and Coordinate Conversion

3.2.1 Vertex Data

3.2.1.1 Vertex Data Format

Each vertex of a polygon rendered on Wii can hold the data shown in Figure 3-2 and Table 3-1.

Figure 3-2 Data that Can be Held by One Vertex

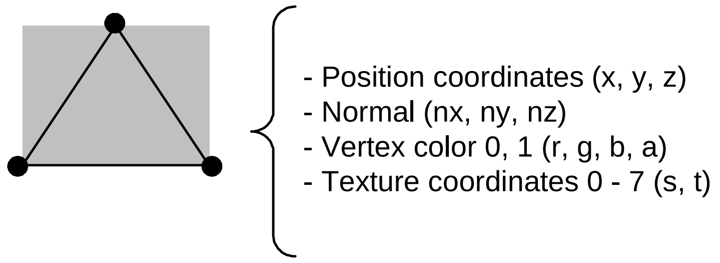

- Position coordinates (x, y, z)
- Normal (nx, ny, nz)
- Vertex color 0, 1 (r, g, b, a)
- Texture coordinates 0 - 7 (s, t)

Table 3-1 Data that Can be Held by One Vertex

Vertex Data	Data Format	Condition
Position coordinates	Fixed-point (8 bits, 16 bits) or floating-point (32 bits)	Required
Normal	Fixed-point (8 bits, 16 bits) or floating-point (32 bits)	Optional
Vertex color 0, 1	Select from RGB565 (16 bits), RGB8 (24bits), RGBX8 (32 bits), RGBA4 (16 bits), RGBA6 (24 bits), and RGBA8 (32 bits)	Optional
Texture coordinates 0 - 7	Fixed-point (8 bits, 16 bits) or floating-point (32 bits)	Optional

Of the vertex data, the elements required for polygon rendering are the matrix for coordinate conversion, and position coordinates. When the coordinate and normal conversion matrices are the same for every vertex of the displayed shape, they do not need to be specified for each vertex. Use of the other vertex data elements is optional.

For each vertex element, a number of data formats can be selected. Fixed-point 16-bit format is half the size of the floating-point format and 8-bit fixed point is one quarter of that size. The precision of coordinates may suffer with fixed-point format, but storing data in this format makes for more efficient use of memory. When fixed-point data is sent to the GPU, it is converted automatically to floating-point format for processing; there is no cost for this conversion process.

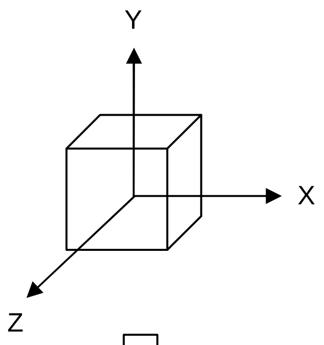
Indices for the matrices used for coordinate, normal, and texture coordinate conversions can also be specified in each vertex.

Wii supports two vertex colors for each vertex, but NintendoWare only supports one vertex color for each vertex.

3.2.2 Conversion of Vertex Position Coordinates

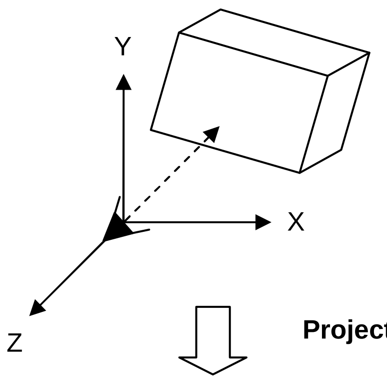
The vertex position coordinates sent to the GPU are model (object) coordinates. As shown in Figure 3-3, model coordinates are converted to visual space coordinates by the modelview matrix, while the visual space coordinates are converted to screen coordinates by the projection matrix and viewport transformation.

Figure 3-3 Coordinate Conversion of Vertex Position Coordinates (For Programmers)



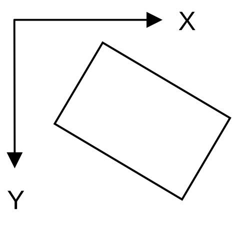
<u>Model (object) Coordinates</u>
The model's reference point is the origin.

Matrix for coordinate conversion (modelview matrix)



<u>Visual Space Coordinates</u>
To derive **world coordinates**, scaling, rotation and translation are applied to the **model coordinates**. World coordinates are then converted to **visual space coordinates** (named so because the viewpoint defines the origin).

Projection matrix, viewpoint transformation



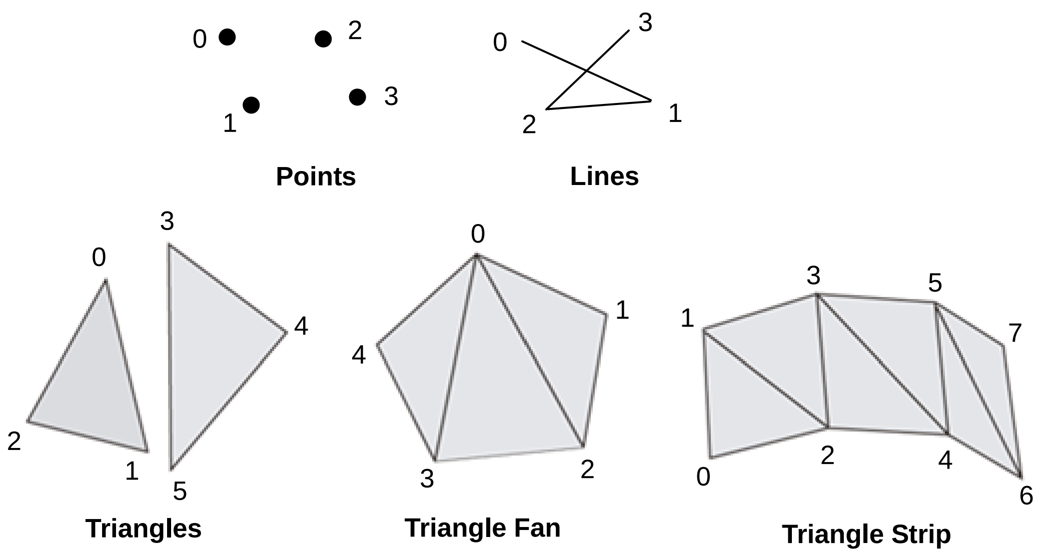
<u>Screen Coordinates</u>
The upper-left corner of the screen is the origin. The back of the screen is the +Z direction.
If the projection matrix is a perspective projection, the image shrinks as the distance from the viewpoint increases.

Generally, up to 10 matrices for coordinate conversion can be loaded into the GPU’s matrix memory. If a coordinate conversion matrix is specified for every vertex in an envelope model and if there are 11 or more matrices, the primitive types have to be partitioned and then rendered.

3.2.3 Primitives Types

Wii can display points, lines, and polygons.

Figure 3-4 Primitive Types



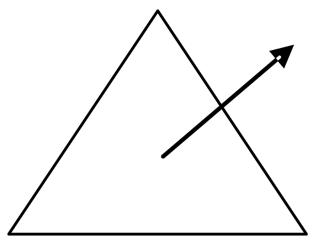
Polygons can be rendered with triangles, triangle fans, triangle strips (see Figure 3-4), and other methods.

Compared to triangles, where one triangular polygon is drawn for every three vertices, triangle fans and triangle strips draw a series of triangular polygons. Reducing the number of process vertices that use triangle fans or triangle strips will reduce the GPU load and data size.

3.2.4 Front Surface of Polygon

As shown in Figure 3-5, on Wii the plane that defines clockwise-aligned vertex data is the front surface of the polygon.

Figure 3-5 Polygon Surfaces



When rendering a polygon, display surfaces can be selected in any of the following combinations.

- Show only the front surface
- Show only the back surface
- Show both surfaces
- Show neither surface

When drawing a closed shape where the back surface of the polygon is not displayed, the unnecessary rendering of the back surface can be eliminated by displaying only the front surface.

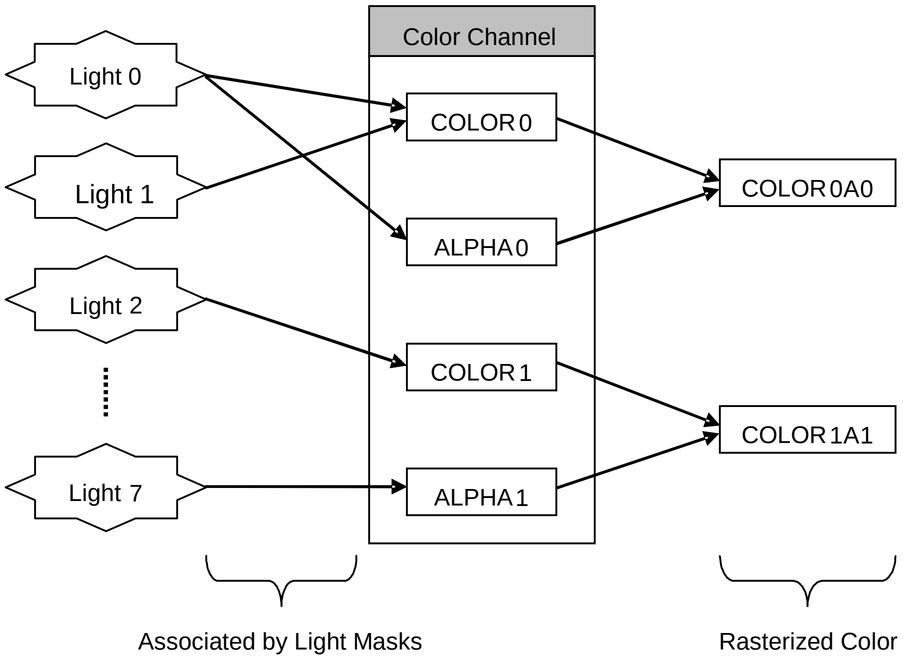
3.3 Shading

3.3.1 Rasterized Color

Wii can use up to eight physical lights. The results of light calculations are stored in as many as four "color channels" and can be used by the TEV as "rasterized colors."

As shown in Figure 3-6, the four color channels are **COLOR0**, **ALPHA0**, **COLOR1**, and **ALPHA1**. COLOR0 and COLOR1 hold the RGB components, while ALPHA0 and ALPHA1 hold only the alpha component. With the TEV, either **COLOR0A0** (color channel 0, combination of COLOR0 and ALPHA0) or **COLOR1A1** (color channel 1, combination of COLOR1 and ALPHA1) can be selected for each stage. For each color channel, the light to be used (light mask), whether to use the material color (alpha) or vertex color (alpha), and the lighting's attenuation function can all be specified.

Figure 3-6 Association of Lights and Color Channels



Wii supports use of different physical lights for diffuse lighting and specular lighting. A total of up to eight diffuse lights and specular lights can be used at a time. Two color channels are used with specular light, (one for diffuse and one for specular lighting), it is necessary to set a separated light mask for each channel.

NintendoWare uses color channel 0 for diffuse lighting and color channel 1 for specular lighting.

3.3.2 Lights

The following attributes can be set for each light.

- Light color (alpha)
- Light position
- Light direction
- Attenuation coefficient based on distance from light to vertex
- Attenuation coefficient based on direction of light and directional angle from light to vertex

By adjusting the coefficients for distance attenuation and angular attenuation different sources of light can be created, including parallel light sources, point light sources, and spotlights (see Figure 3-7 and Figure 3-8). (A parallel light source approximates a point light source when positioned far away from an object with no attenuation.)

Figure 3-7 Types of Light

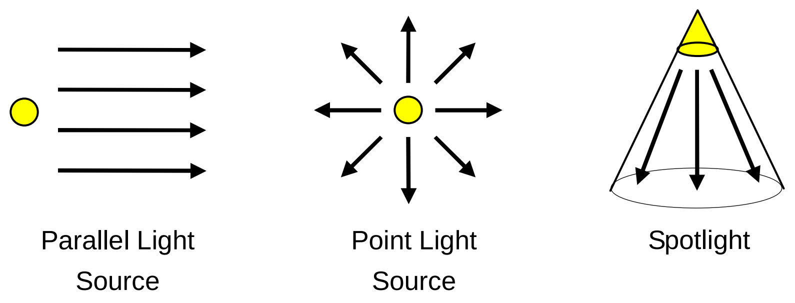


Figure 3-8 Formulas for Calculating Light (For Programmers)

$$\boxed{\text{Rasterized Color} = \text{Material x LightFunc}}$$

$$\text{Material} = \text{Material Source} = \begin{cases} \text{Material Color Register} \\ \text{Vertex Color} \end{cases}$$

$$\text{LightFunc} = \begin{cases} \text{1.0 (No Light Calculation)} \\ \text{Clamp}\left[\text{Ambient} + \sum_{i=0}^{7} \left(\text{LightMask(i) * Attn(i) * DiffuseAttn(i) * Color(i)}\right)\right] \end{cases}$$

$$\text{Ambient} = \text{Ambient Source} = \begin{cases} \text{Ambient Color Register} \\ \text{Vertex Color} \end{cases}$$

LightMask(i) = Light mask (1 if light i enabled; 0 if disabled)

Attn (i) = Distance attenuation and angular attenuation (or specular attenuation)

DiffuseAttn (i) = Diffuse attenuation
(Calculated from scalar product of normal and vector from vertex to light i)

Color (i) = The light color (alpha) of light i

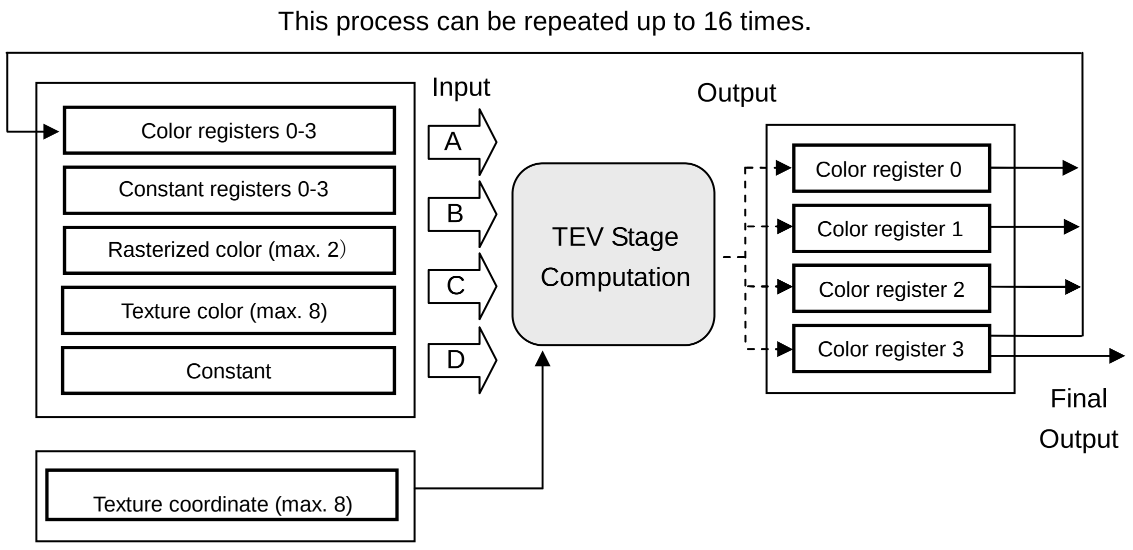
3.4 TEV (Texture Environment Unit)

3.4.1 What is the TEV?

The TEV is the mechanism that performs rendering by using input sources (rasterized color, texture color, color registers, and so on) and ultimately determines the final pixel color. Many material effects can be attained with the TEV, by mixing rasterized color and texture color, blending a number of textures, and applying color to polygons using the color registers.

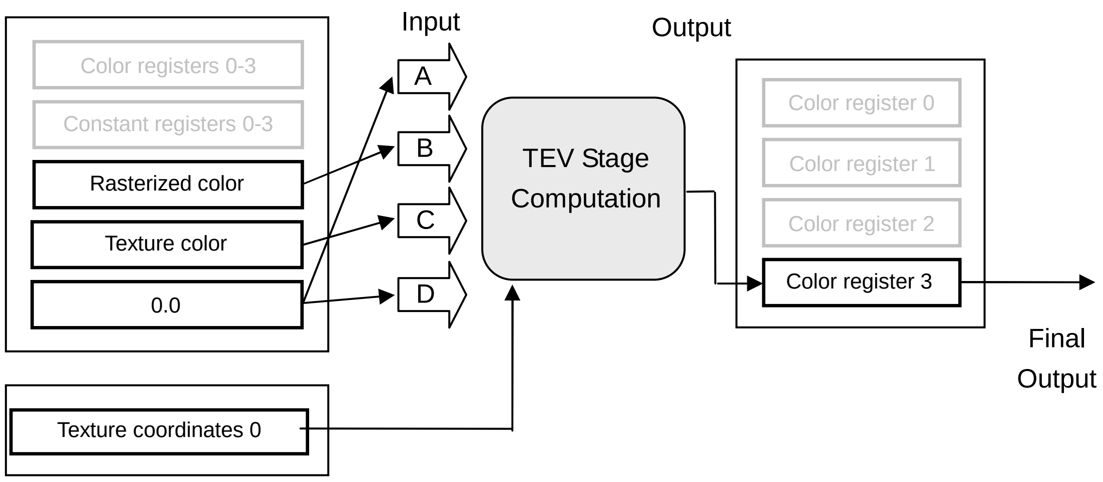
The TEV can perform the computational expression described in Figure 3-9 from one to sixteen times. A single execution of the calculation is called a stage. The first stage is called "stage 0" and computations can be performed up to stage 15. For each stage of computation, four inputs (A, B, C, and D) are selected from the various input sources.

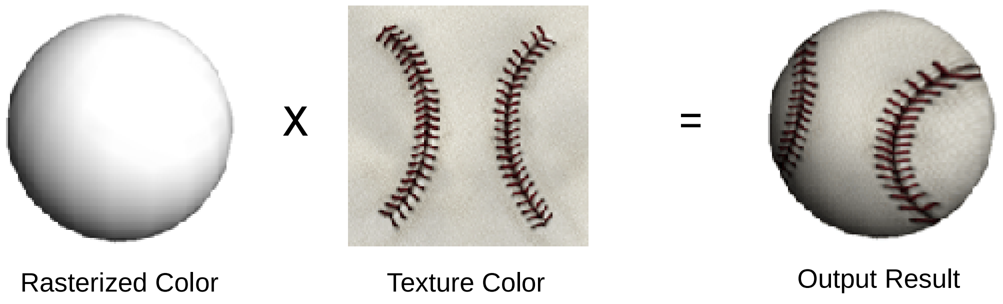
Figure 3-9 TEV Block Diagram



For example, the process is performed in one stage when a texture is mixed with a rasterized color, as shown in Figure 3-10.

Figure 3-10 Example of Output when Combining Texture with Rasterized Color





Only one texture can be input for a given stage. So at least two stages are needed to combine two textures.

3.4.2 Registers that Can be Used with TEV Stage Computations

Before explaining the TEV stage computation, the registers used in the computations must be described.

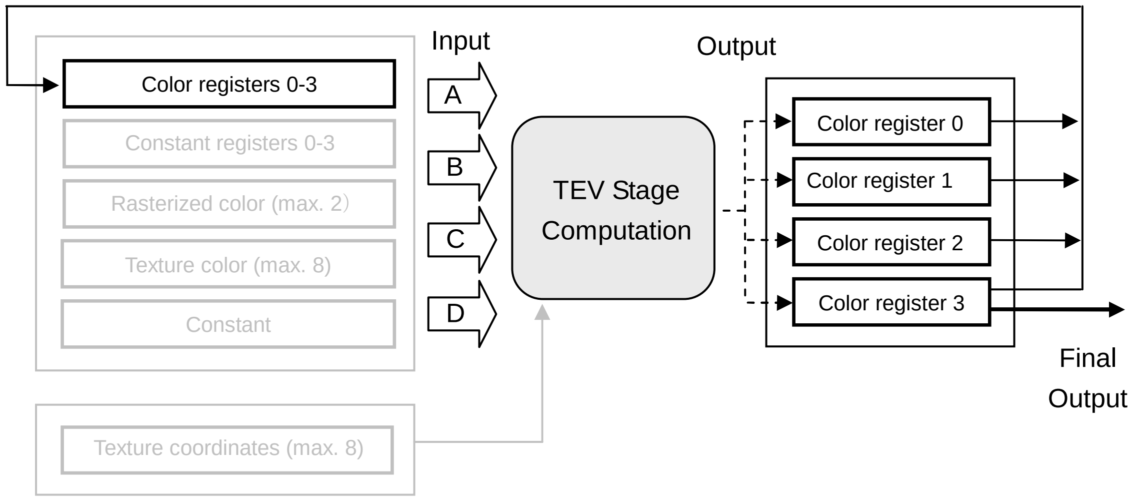
Four color registers and four constant registers can be shared for computations in all 16 TEV stages. The features of each register are described below.

3.4.2.1 Color Registers

For color registers to serve as input sources for stage computations, their colors or values can be set (see Figure 3-11). Color registers can also be used as destinations that store the results of the stage computations.

Figure 3-11 Positioning Color Registers

This process can be repeated up to 16 times.



Each color register comprises four RGBA color components (red, green, blue, alpha), and each of these components can be set to an integer value between -1024 and 1023 (see Figure 3-12). In terms of the standard interpretation of color components (which range from 0.0 to 1.0), "0" corresponds to 0.0 and "255" to 1.0. If the result of the stage computation is smaller than 0 or larger than 255, it can still be stored in the color registers.

Figure 3-12 Features and Uses of Color Registers

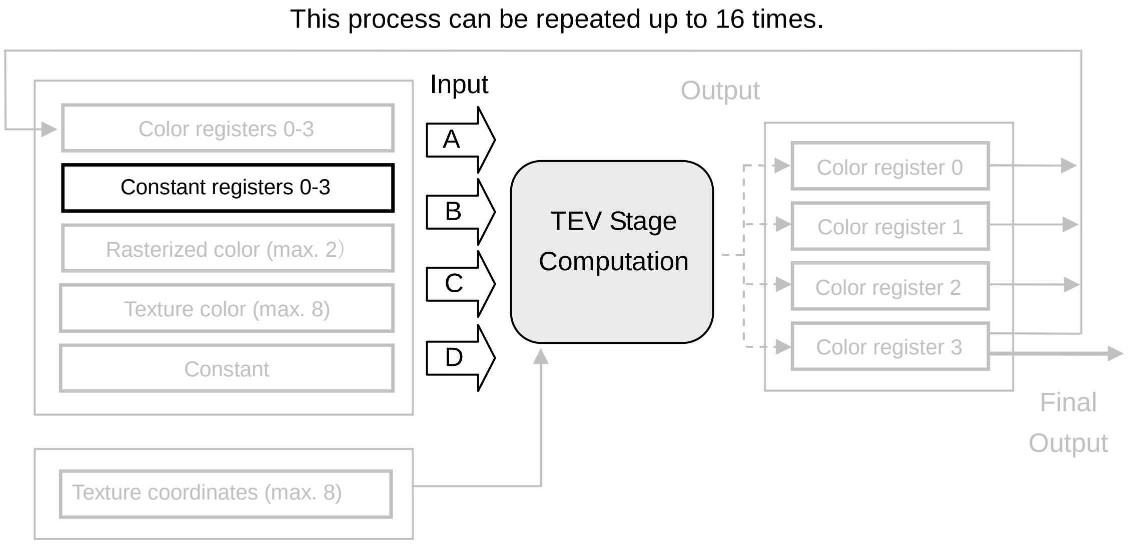
Color register 0
Color register 1
Color register 2
Color register 3

- Each RGBA component is stored as an integer value between -1024 and 1023
- Multiple registers can be used simultaneously in one stage
- Results of stage computation are invariably stored in one of the color registers
- Results of stage computation can be used in subsequent stages
- For the final stage, output must be to color register 3

3.4.2.2 Constant Registers

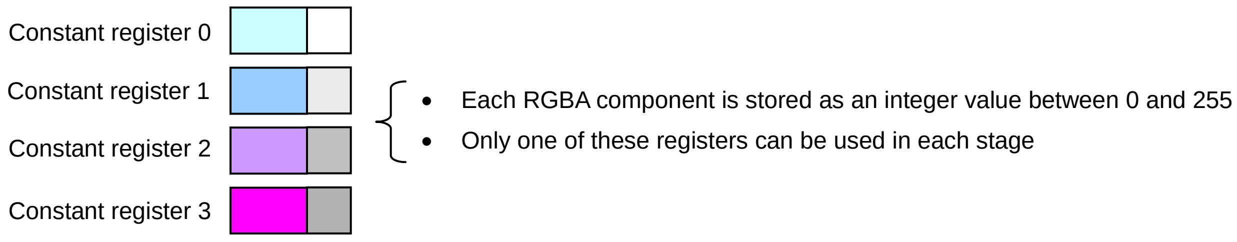
To use constant registers with stage computations, set their colors or values (see Figure 3-13).

Figure 3-13 Positioning of Constant Registers



Similar to the color registers, each constant register comprises four RGBA color components (see Figure 3-14). However, each component can be set to integer values between 0 and 255. In terms of the standard interpretation of color components (which range from 0.0 to 1.0), "0" corresponds to 0.0 and "255" to 1.0.

Figure 3-14 Features and Uses of Constant Registers


Constant register 0
Constant register 1
Constant register 2
Constant register 3
• Each RGBA component is stored as an integer value between 0 and 255
• Only one of these registers can be used in each stage

3.4.3 TEV Stage Computations

3.4.3.1 Color Stage and Alpha Stage

As shown in Figure 3-15, each stage of TEV computation consists of two components, computation of color (the **color stage**) and computation of alpha (the **alpha stage**). Each component stage is independent and can be configured separately. In addition, either **TEV computation expression** or **compare mode** can be selected for computation in every stage.

Figure 3-15 Color and Alpha Stages

$$\begin{array}{c}\text{TEV Stage}\\ \text{Computation}\end{array}\begin{cases}\text{Color Stage Computation: TEV Computation Expression or Compare Mode}\\ \\ \text{Alpha Stage Computation: TEV Computation Expression or Compare Mode}\end{cases}$$

The same number of stages is set for color stage and alpha stage. Therefore, three alpha stages (not one) must be used with three color stages.

3.4.3.2 TEV Computation Expression

When choosing a TEV computation expression for stage computation, use the expression shown in Figure 3-16.

Figure 3-16 TEV Computation Expression

$$\left[\boxed{D}\underset{(1)}{\boxed{\begin{array}{c}+\\ \\ -\end{array}}}\{(\;1\;-\boxed{C})\times\boxed{A}+\boxed{C}\times\boxed{B}\}\overset{\text{Bias}}{\underset{(2)}{\boxed{\begin{array}{c}-0.5\\+0.0\\+0.5\end{array}}}}\right]\times\overset{\text{Scale}}{\underset{(3)}{\boxed{\begin{array}{c}0.5\\1.0\\2.0\\4.0\end{array}}}}$$

Select one input source for each of A, B, C, and D. Note that the color stages and alpha stages take different kinds of input sources. (For a more detailed explanation, see Chapter 4 Examples of TEV Settings.)

For term (1) of the expression, use either plus (+) or minus (–) to add to or subtract from (D). For term (2), select one of the values -0.5, +0.0, or +0.5 to increase or decrease the overall brightness of color stage or the transparency of alpha stage. For term (3), select one of the values 0.5, 1.0, 2.0, or 4.0. A setting of 0.5 will halve the brightness or transparency obtained from the prior computation, while a setting of 2.0 will double it.

3.4.3.3 Compare Mode

To calculate pixel color, the TEV computation expression uses the input sources A, B, C, and D. As an alternative to the expression, perform output processing after the conditional expressions shown in Figure 3-17. In this **compare mode**, conditional branching can be performed using A, B, C, and D.

Figure 3-17 Compare Mode

$\boxed{\text{If A = B, then output D + C, otherwise output D}}$

or

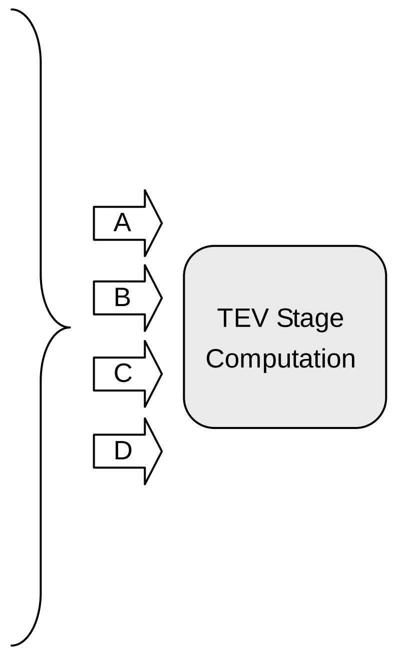
$\boxed{\text{If A > B, then output D + C, otherwise output D}}$

3.4.3.4 Color Stage

For the color stage, any one of the input sources listed in Figure 3-18 can be set for each, A, B, C and D.

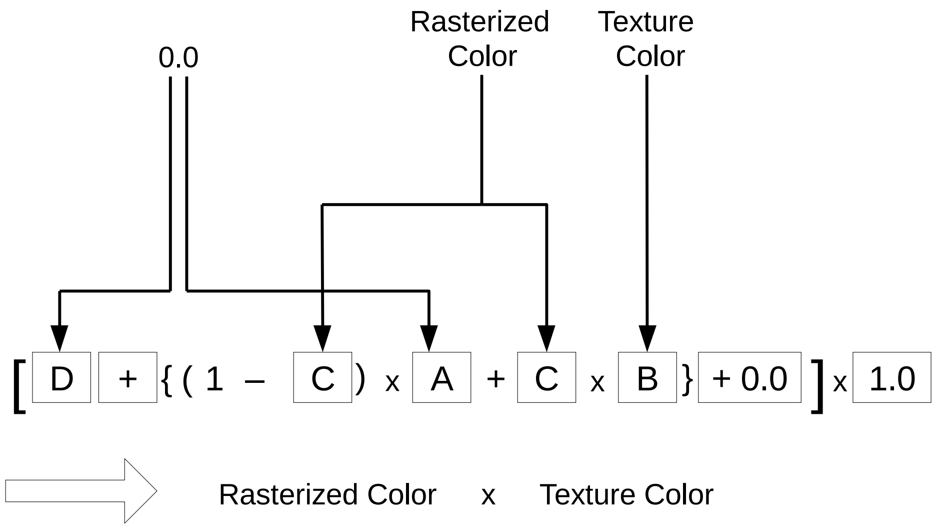
Figure 3-18 Color Stage and Input Sources

- Rasterized color (RGB)
- Rasterized alpha
- Texture color (RGB)
- Texture alpha
- Color (RGB) of color register 0
- Alpha of color register 0
- Color (RGB) of color register 1
- Alpha of color register 1
- Color (RGB) of color register 2
- Alpha of color register 2
- Color (RGB) of color register 3
- Alpha of color register 3
- Constant (1.0, 0.5 or 0.0)
- Value specified by constant register
  - ➢ One of the constant registers (0, 1, 2, or 3)
  - ➢ One of the register components (RGB, R, G, B, or A)



To use a constant register as an input source, specify the constant register (between 0 and 3) and the RGBA component (R, G, B, or A). For example, to paste a texture on a shaded polygon, the rasterized color is set in C and the texture color is set in B. Then they are multiplied in the TEV computation expression, as shown in Figure 3-19.

Figure 3-19 Example of Stage Settings for Applying Shadows to Textures


Rasterized Color
Texture Color
0.0
[D + { (1 – C) x A + C x B } + 0.0] x 1.0
Rasterized Color x Texture Color

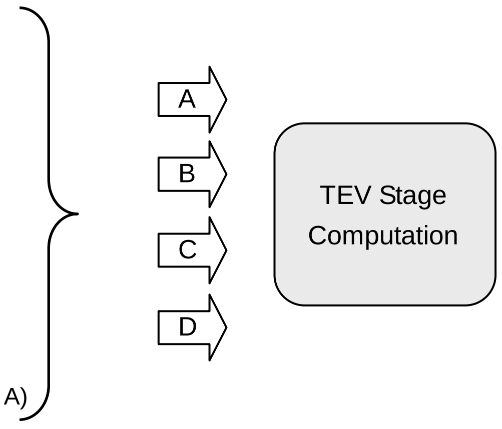
For detailed examples of TEV stage settings, see Chapter 4 Examples of TEV Settings.

3.4.3.5 Alpha Stage

For the alpha stage, any one of the input sources shown in Figure 3-20 can be set for each of A, B, C, and D.

Figure 3-20 Alpha Stage and Input Sources

- Rasterized alpha
- Texture alpha
- Alpha of color register 0
- Alpha of color register 1
- Alpha of color register 2
- Alpha of color register 3
- 0.0
- Value specified by constant register
    - ➢ One of the constant registers (0, 1, 2, or 3)
    - ➢ One of the constant register components (R, G, B, or A)



To use a constant register as an input source, specify the constant register (between 0 and 3) and the component (R, G, B, or A). For detailed examples of TEV stage settings, see section 4 Examples of TEV Settings.

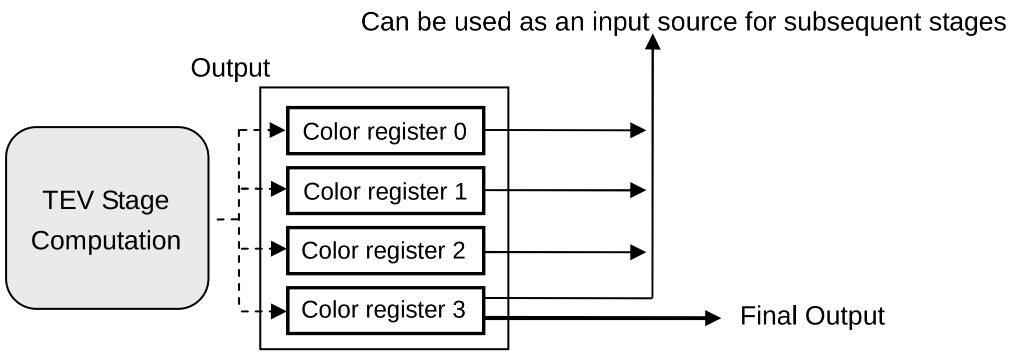
3.4.3.6 Outputting the Results of Stage Computation

As shown in Figure 3-21, the result of each stage computation for color stages must be output to the RGB of one color register (0 through 3). The result of each stage computation for alpha stages must be output to the alpha of one color register (0 through 3).

If multiple stages are used, the results of intermediate stage computations are temporarily held in a color register and can be used in subsequent stages.

Regardless of the number of stages used, the final stage must be output to color register 3.

Figure 3-21 Output of the Stage Computation Result

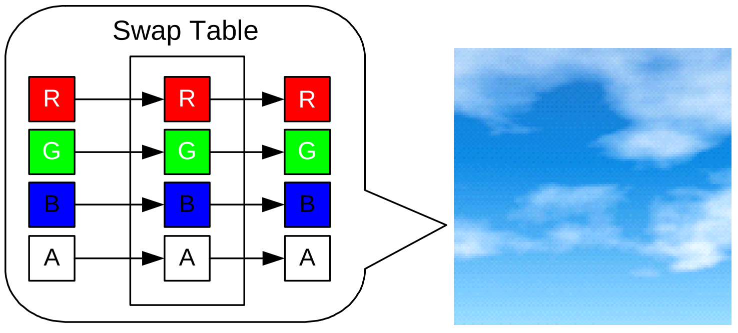


3.4.4 Swap Table

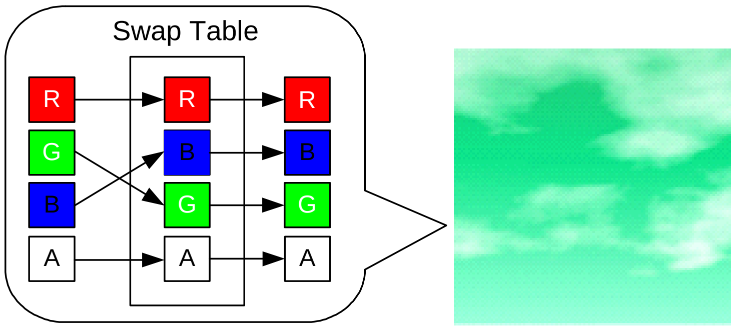
The **swap feature** (see Figure 3-22) allows the interpretation of the RGBA components to be changed before using rasterized colors and texture colors as input sources for various TEV stages. With the **swap table**, four swap change patterns can be simultaneously registered per process, for a series of TEV settings.

Normally a swap table for which the RGBA components are not altered would be used. However, by switching the referenced swap tables or by changing the swap table settings, effects such as showing one texture in a different color can be created. Because the swap feature is always executed, changing the settings does not change the processing load.

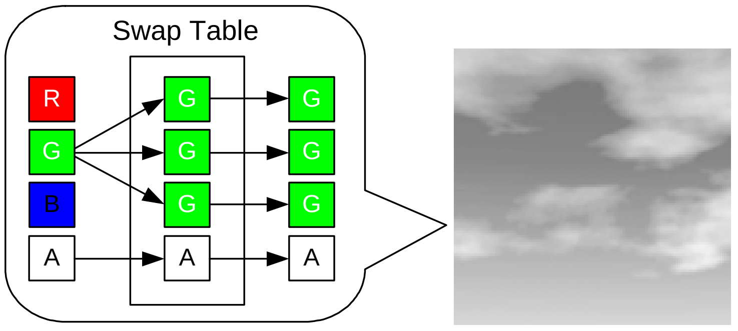
Figure 3-22 The Swap Feature



Texture color is used as is.



Green and blue components are swapped.
Blue parts of the texture become green.



Green component is also used for red and blue components. The texture is shown in grayscale.

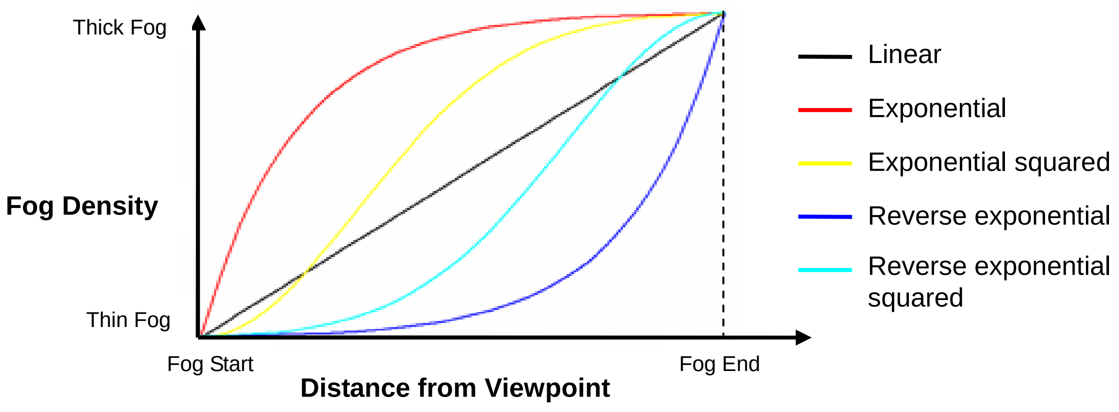
3.4.5 Fog

Fog color can be blended with the pixel color that is output from the final active TEV stage. The amount of blending with the fog color is determined by the fog function, which is used for the viewpoint-to-pixel distance. Table 3-2 lists the types of fog functions, while Figure 3-23 shows the fog function graph.

Table 3-2 Types of Fog Function

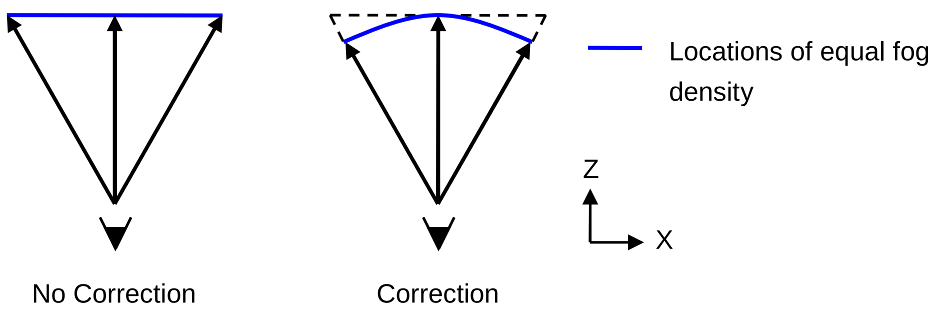
Type	How Fog is Applied
Linear	Applied evenly toward the rear.
Exponential	Applied abruptly up front.
Exponential squared	Applied up front.
Reverse exponential	Applied abruptly in rear, not up front.
Reverse exponential squared	Applied in rear, not up front.

Figure 3-23 Fog Function Graph



To increase the fog density at screen edges, enable correction of the fog range (see Figure 3-24). This correction is only possible in the x-direction of the screen. The fog in the y-direction is not corrected.

Figure 3-24 Correction of Fog Range



3.5 PE (Pixel Engine)

3.5.1 What is the PE?

Z-comparisons, blending, and dithering are the final steps performed in the pixel pipeline for output to the frame buffer. These processes are collectively called the Pixel Engine (PE).

In the Z comparison process, Z values are compared to determine if the pixel scheduled for writing to the frame buffer should be rasterized. In the blending process, the rasterized pixel color is blended with the color that is in the frame buffer. Dithering is performed as the final process.

This section describes each of these three processes.

3.5.2 Z Comparison

To perform a conditional write to the frame buffer, compare the Z value of the pixel to be written to the frame buffer with the Z value of the pixel already in the buffer (the Z buffer's Z value) see Table 3-3.

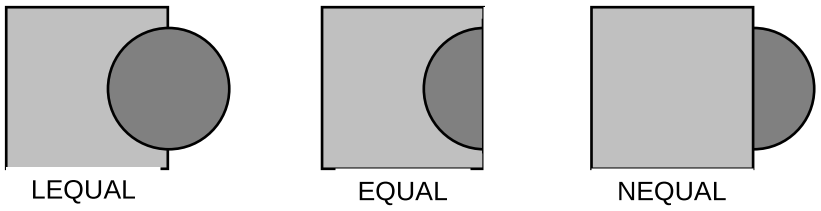
Table 3-3 Types of Z Comparison Expressions

Comparison Expression Types	Description
NEVER	Never write.
LESS	Write if new pixel's Z value < the Z buffer's Z value.
LEQUAL	Write if new pixel's Z value ≤ the Z buffer's Z value.
EQUAL	Write if new pixel's Z value = the Z buffer's Z value.
NEQUAL	Write if new pixel's Z value ≠ the Z buffer's Z value.
GEQUALL	Write if new pixel's Z value ≥ the Z buffer's Z value.
GREATER	Write if new pixel's Z value > the Z buffer's Z value.
ALWAYS	Always write.

Normally, a new pixel is written to the frame buffer if it is closer to the viewpoint (that is, if it has a smaller Z value), and LEQUAL is used. Use the EQUAL expression to write only the overlapping parts of polygons. Note that due to the Z buffer's precision and the distance from the viewpoint, errors my cause flicker.

Figure 3-25 shows the result of Z comparison for a circular polygon drawn in the same place after a square polygon.

Figure 3-25 Examples of Z Comparison

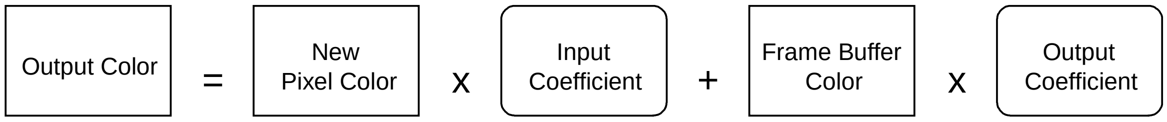


Z comparison can be done before or after texture processing. Better performance is obtained by performing the Z comparison first, since the texture processing is not performed on the pixels that are not displayed. However, when using alpha-comparison, texture processing must precede Z comparison.

3.5.3 Blending

To blend the color of a pixel that is to be written to the frame buffer with the frame buffer color, use the expression in Figure 3-26.

Figure 3-26 Expression for Blending



Adjustment of the input and output coefficients (see Table 3-4) allows calculations such as mixing (translucence), adding, and multiplying to be performed.

Table 3-4 Examples of Blending Coefficients

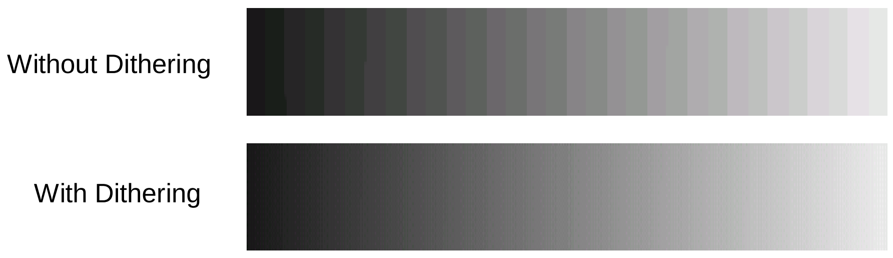
Operation	Input Coefficient	Output Coefficient
Mix (translucent)	Alpha of new pixel	1 (Alpha of new pixel)
Add	Alpha of new pixel	1
Multiply	Color of frame buffer	0

In addition to blending, the new pixel's color can be subtracted from the frame buffer's color, and logical operations such as AND and OR can be performed between a new pixel’s color and frame buffer’s color. If the frame buffer supports alpha, the same (blending, subtraction, and logical) processes can be performed on alpha values.

3.5.4 Dithering

If the frame buffer's pixel format is RGB565 or RGBA6, pixels can be dithered after blending. As illustrated in Figure 3-27, dithering creates a smoother transition of coloring, reducing the presence of boundaries between levels.

Figure 3-27 Dithering



3.6 Video Output

3.6.1 Frame Buffer

The frame buffer in graphics memory (the Embedded Frame Buffer, or **EFB**) is first copied to the external frame buffer in main memory (the eXternal Frame Buffer, or **XFB**) and then sent to the video interface. During EFB-to-XFB copying, antialias filtering, deflicker filtering, gamma correction, RGB-to-YUV conversion, and Y-scaling are performed.

Table 3-5 shows the pixel formats that can be used by the EFB.

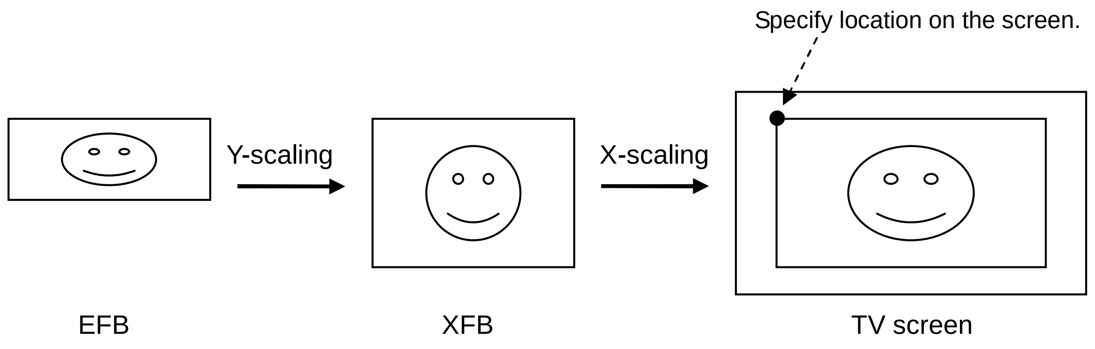
Table 3-5 Pixel Formats of the EFB

Pixel Format	Number of RGB (A) Bits	Number of Z Bits	Antialiasing
`RGB8_Z24`	8 bits for each RGB	24 bits	Not possible
`RGBA6_Z24`	6 bits for each RGBA	24 bits	Not possible
`RGB565_Z16`	R - 5 bits, G - 6 bits, B - 5 bits	16 bits	Possible

The maximum resolution for the EFB is 640 × 528 without antialiasing and 640 × 264 with antialiasing. The maximum screen space on the TV screen is 720 × 480 for NTSC or MPAL, and 720 × 574 for PAL.

To display to a screen space that is larger than the maximum EFB resolution or to change the aspect ratio (for example, to fit a widescreen TV), use Y-scaling for the vertical direction when copying data from the EFB to the XFB and X-scaling for horizontal direction when outputting from the XFB to video (see Figure 3-28). To increase vertical resolution, the EFB can also be rendered in several steps and copied to an XFB that is larger than the EFB.

Figure 3-28 Scaling Frame Buffer

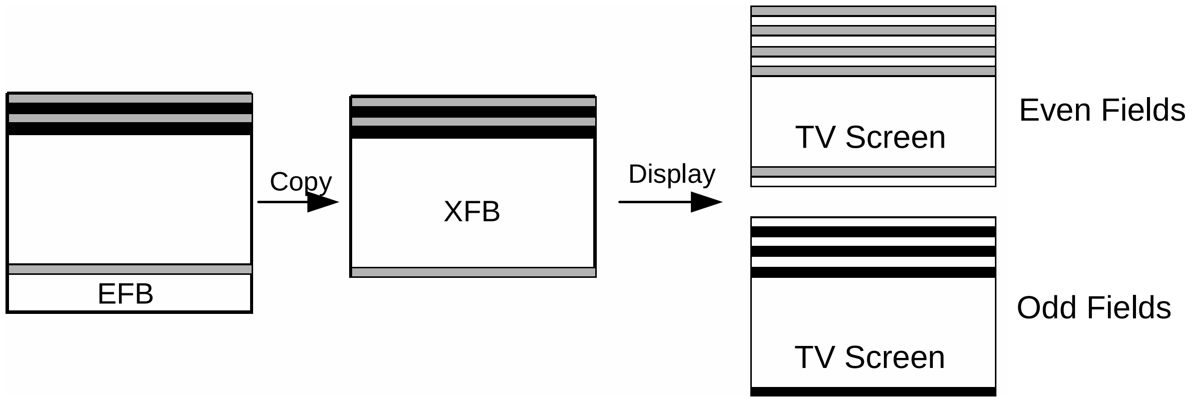


For each TV format (NTSC, MPAL, or PAL), select one of the following rendering modes.

- Interlace
- Interlace (field rendering)
- Noninterlace (not recommended)
- Progressive

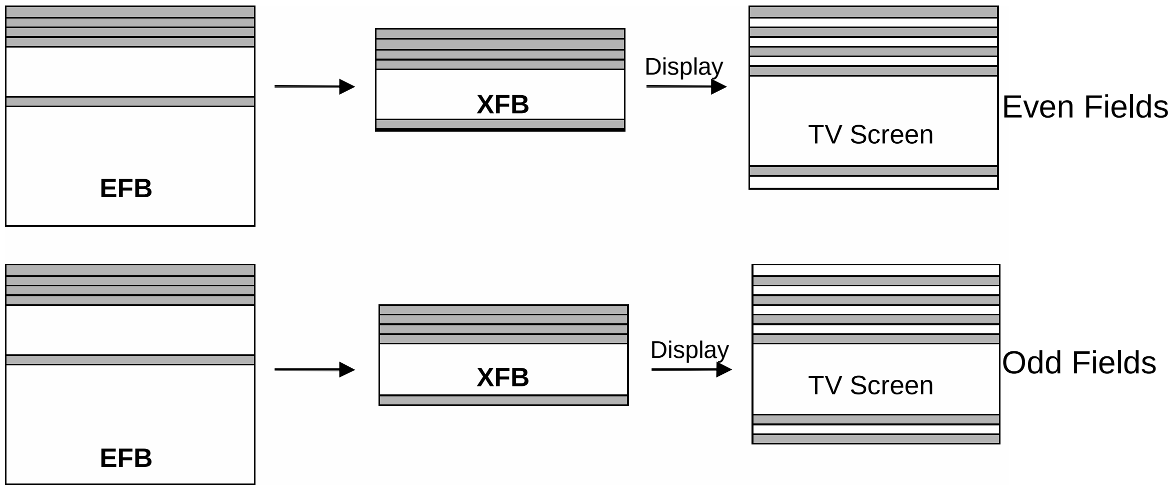
In interlace mode (see Figure 3-29), a scan is done twice for each screen display (once for odd-numbered fields and once for even-numbered fields). All fields are rendered, then each subsequent frame is displayed by alternately scanning the odd and even fields.

Figure 3-29 Interlace Mode



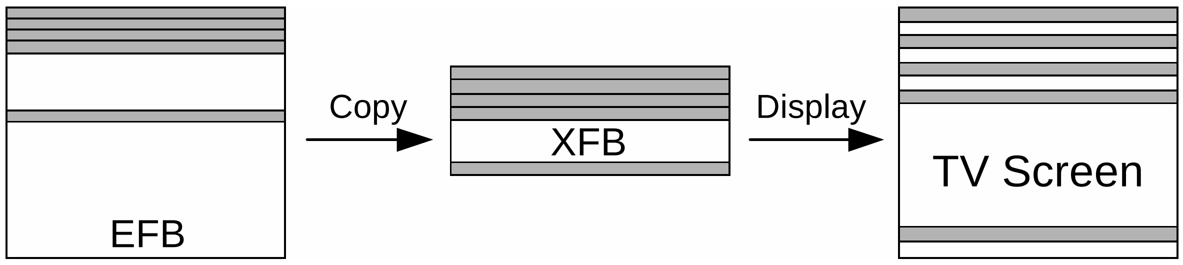
In interlace (field rendering) mode (see Figure 3-30), each frame is displayed by alternately rendering odd and even fields. Compared to the regular interlace mode, the EFB is only half the size, and the processing load is lighter. However, if a process is dropped, the screen from the previous frame will appear and one scan line of jitter will be visible on the screen.

Figure 3-30 Interlace (Field Rendering) Mode



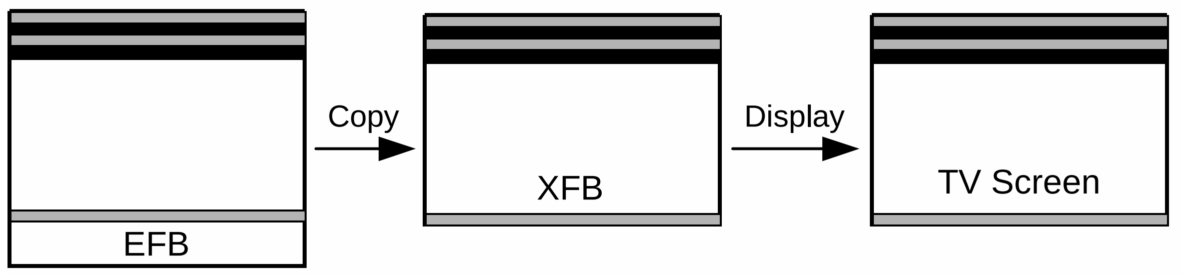
In noninterlace mode (see Figure 3-31), the even fields are displayed in each frame only after these fields are rendered. Compared to the regular interlace mode, the EFB is only half the size and the processing load is lighter. However the vertical resolution is also reduced by half.

Figure 3-31 Noninterlace Mode



In progressive mode (see Figure 3-32), all fields are displayed in each frame after they are rendered. This mode provides high picture quality and has none of the flicker of the interlace modes, but it can only display on a television that supports progressive mode.

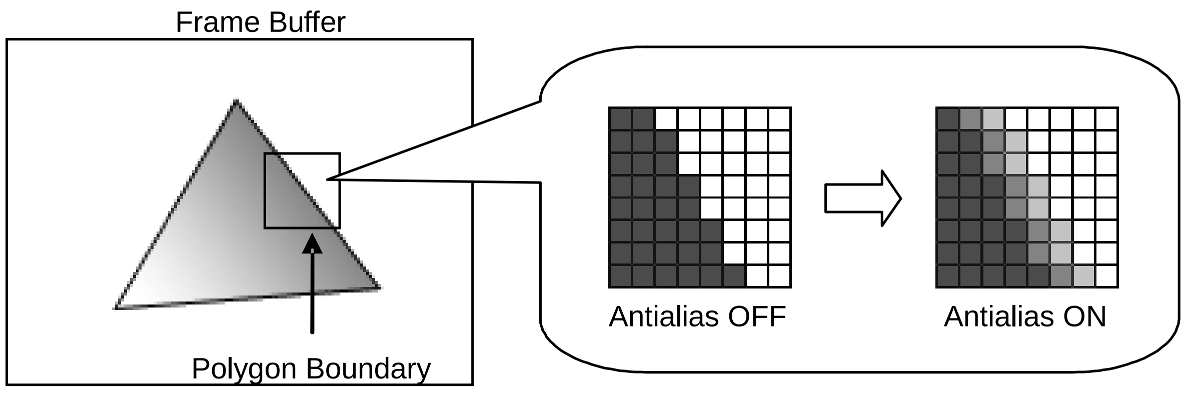
Figure 3-32 Progressive Mode



3.6.2 Antialiasing

Antialiasing is a feature that smoothes the edges of rendered polygons and lines by mixing the colors of the pixels above and below each boundary pixel, thus reducing aliasing artifacts (jaggies). Antialiasing can only be used when the frame buffer format is RGB565_Z16 and the resolution is 640 x 264 or lower. To apply antialiasing to an image of 640 x 480 resolution, perform the same rendering process twice (render the upper and lower halves separately) then combine them. Figure 3-33 is only a conceptual diagram for antialiasing; the actual display may vary.

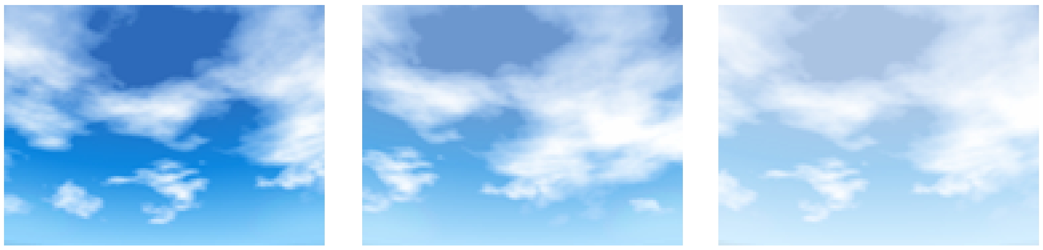
Figure 3-33 Antialias



3.6.3 Gamma Correction

Select 1.0, 1.7, or 2.2 for the gamma correction value. The picture being rendered in the frame buffer is not changed. However, when the picture is output to video, the overall screen will appear lighter with a larger gamma correction value. Figure 3-34 only shows gamma correction; the actual display may vary.

Figure 3-34 Gamma Correction



Gamma = 1.0 Gamma = 1.7 Gamma = 2.2

3.6.4 Deflicker

The deflicker filter is a feature for reducing flicker when interlace mode is used. The deflicker filter operates on the entire screen.

3.6.5 Converting From RGB to YUV

When data is copied from the EFB to the XFB, the pixel format is converted from RGB to YUV. This conversion process reduces the data volume to two-thirds of its original size, allowing space to be saved in the XFB.

The YUV format expresses colors by their brightness (Y), blue color component (U), and red color component (V). The human eye has a relatively low sensitivity to colors, so a small number of bits can be allocated for the U and V values.

4 Examples of TEV Settings

Using the TEV, Wii can express a variety of materials. In this chapter, different examples of TEV settings are used to demonstrate how to express basic materials.

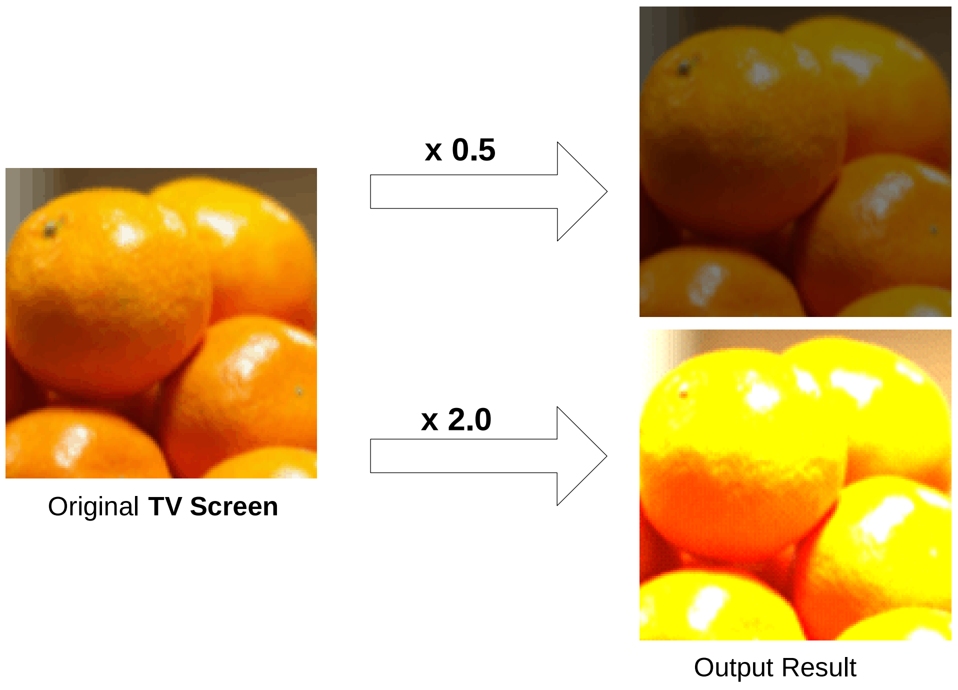
- 4.1 Modulation
- 4.2 Blending Using Two Textures
  - 4.2.1 Addition
  - 4.2.2 Subtraction
  - 4.2.3 Multiplication
  - 4.2.4 Addition and Multiplication Combined
  - 4.2.5 Decal
  - 4.2.6 Decal (Applied)
  - 4.2.7 Proportional Blending
- 4.3 Two-Color Interpolation

4.1 Modulation

4.1.1 Features of Modulation

The brightness of the TEV computation expression output result can be halved (darkened), doubled, or quadrupled as shown in Figure 4-1.

Figure 4-1 Modulated Images



4.1.2 What is Modulation?

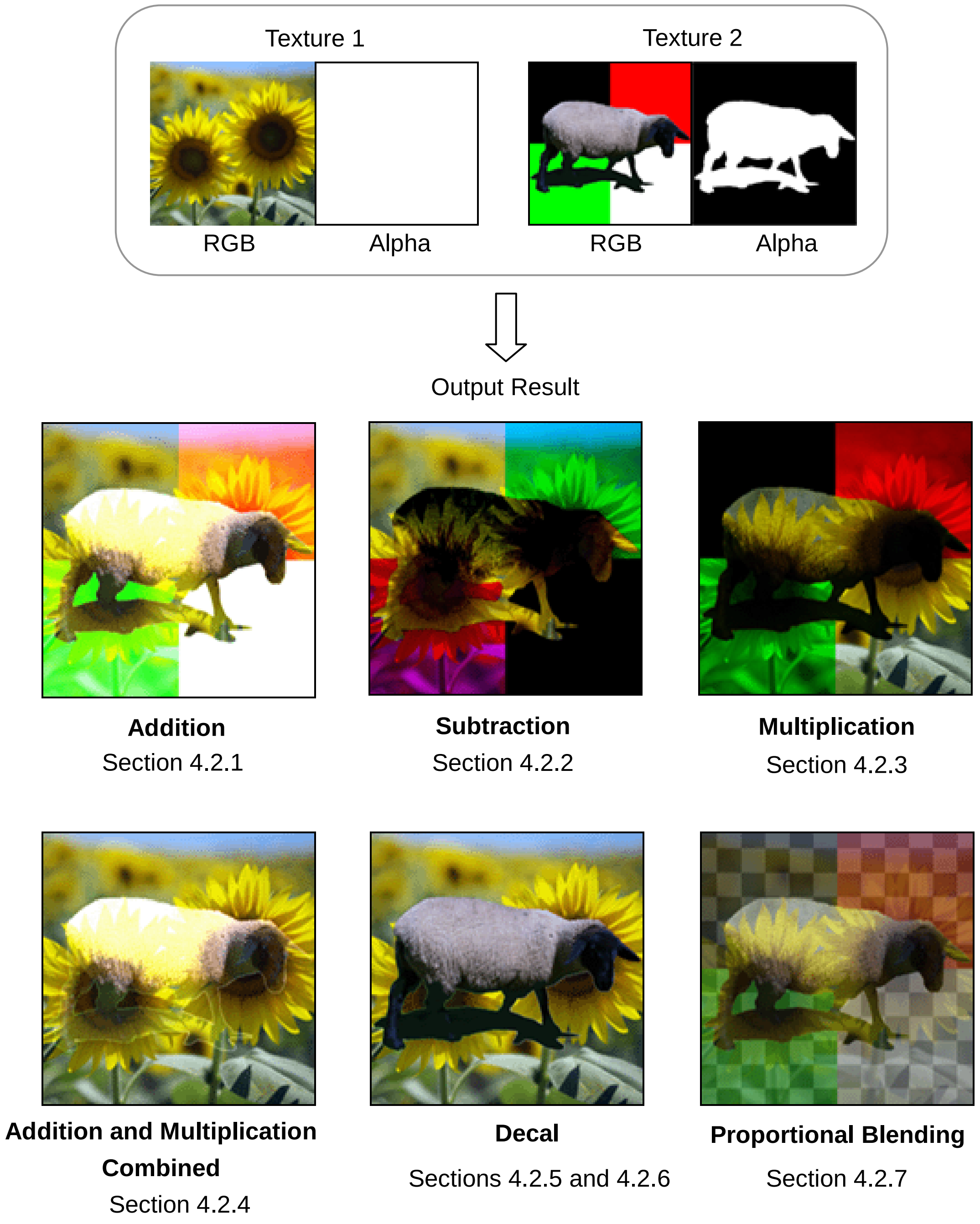
Modulation uses the **scale** value located on the far right of the TEV computation expression. Scale is specified by 0.5, 1.0, 2.0, or 4.0. Normally, 1.0 is used.

$$\text{Result} = [\, D + \{ (1 - C) \times A + C \times B \} + 0.0 \,] \times \textbf{Scale}$$

4.2 Blending Using Two Textures

Various blending effects can be expressed by pasting two layered textures. The basic blending effects and their configuration are explained separately (see references listed in Figure 4-2). The grid pattern visible in the output result of Figure 4-2 indicates transparency due to alpha.

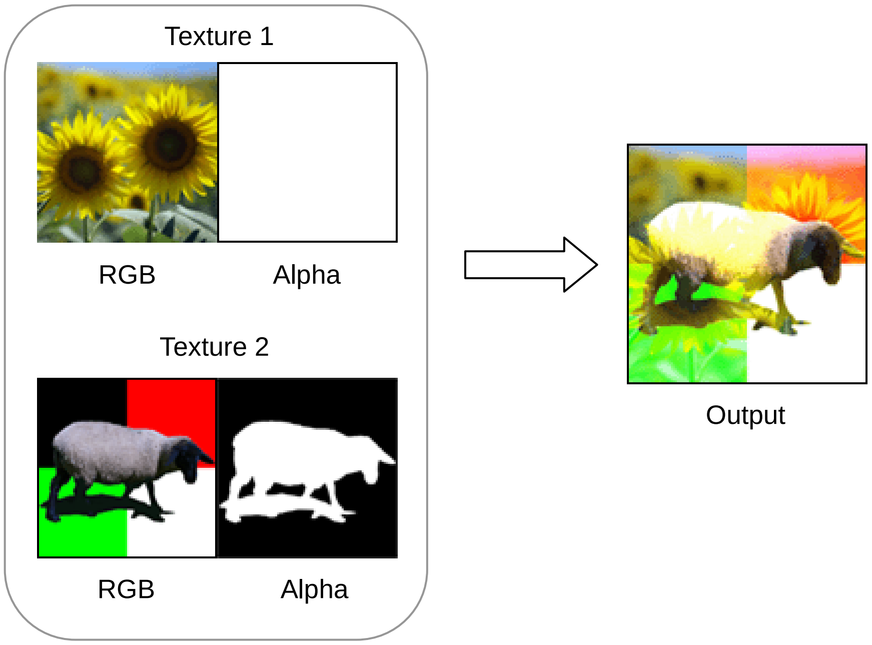
Figure 4-2 Blending of Two Textures



4.2.1 Addition

This method adds one texture to another, brightening the added portions of the two textures (see Figure 4-3). Addition can also be used for effects such as light and white fadeout.

Figure 4-3 Example of Texture Addition



4.2.1.1 How Addition Works

In the TEV computation expression, where 0.0 is specified for **B** and **C**, **D + A** is used:

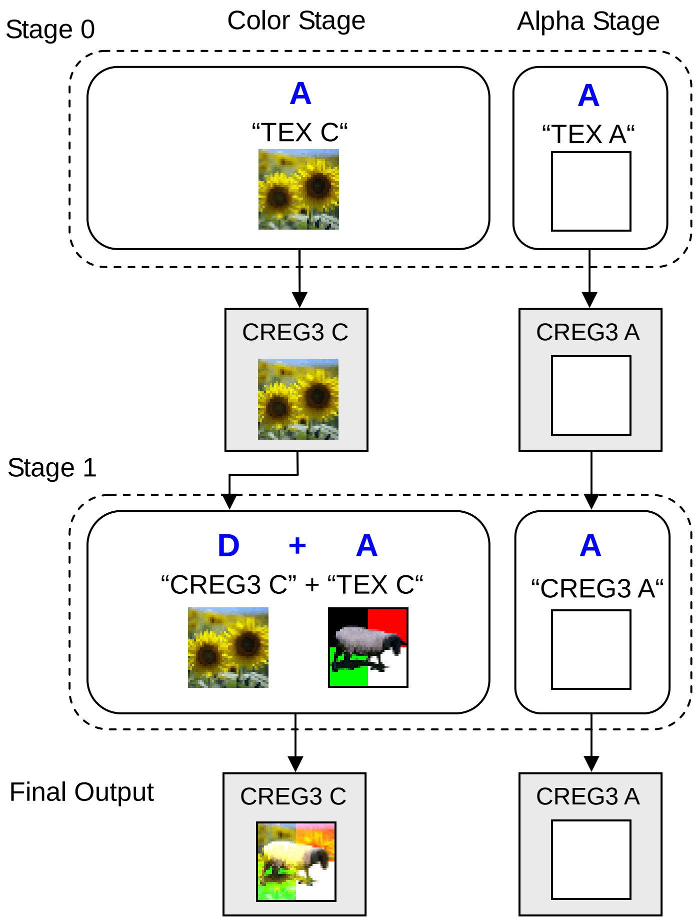
$$\begin{aligned}\text{Result} &= [\, \mathbf{D} + \{ (1 - C) \times \mathbf{A} + C \times B \} + 0.0 \,] \times 1.0 \\ &= [\, \mathbf{D} + \{ (1 - 0.0) \times \mathbf{A} + 0.0 \times 0.0 \} + 0.0 \,] \times 1.0 \\ &= \mathbf{D} + \mathbf{A}\end{aligned}$$

4.2.1.2 Stage Settings for Adding Textures

Addition of two textures requires the use of two stages of the TEV. In stage 0 (see Figure 4-4), the texture color and alpha are output unmodified to color register 3 by both the color and alpha stages. Next, **D + A** from stage 1 is used to add the second texture to the result of the previous stage.

In Figure 4-4, TEX C refers to the texture's RGB components and TEX A to its alpha component. CREG*C refers to the color register's RGB components, and CREG* A to its alpha component.

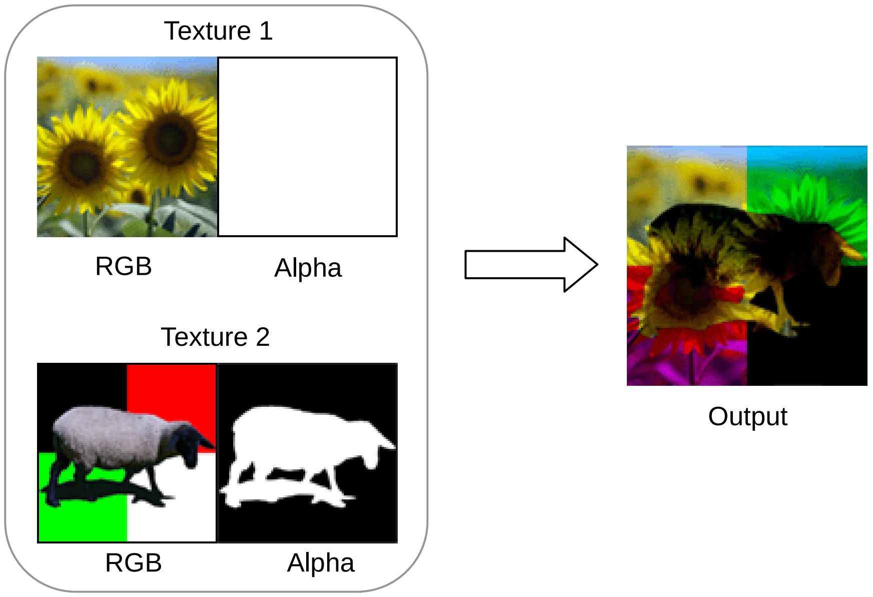
Figure 4-4 Example of Stage Settings for Addition



4.2.2 Subtraction

This method subtracts one texture from another, making the subtracted portions darker (see Figure 4-5). Subtraction can also be used for the effects such as fadeout.

Figure 4-5 Example of Subtraction



4.2.2.1 How Subtraction Works

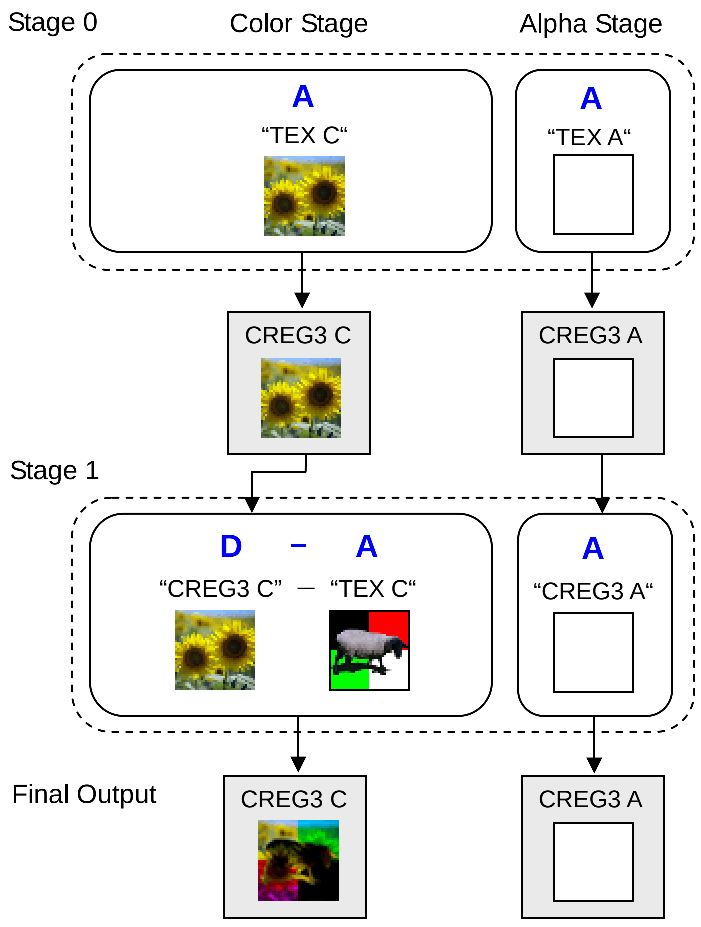
In the TEV computation expression, where 0.0 is specified for **B** and **C**, **D** – **A** is used:

$$\begin{aligned}\text{Result} &= [\,\mathbf{D} - \{(1 - C) \times \mathbf{A} + C \times B\} + 0.0\,] \times 1.0 \\ &= [\,\mathbf{D} - \{(1 - 0.0) \times \mathbf{A} + 0.0 \times 0.0\} + 0.0\,] \times 1.0 \\ &= \mathbf{D} - \mathbf{A}\end{aligned}$$

4.2.2.2 Stage Settings for Subtraction

Subtraction of two textures requires the use of two stages of the TEV. In stage 0 (see Figure 4-6), the texture color and alpha are output unmodified to color register 3 by both the color and alpha stages. Next, **D** – **A** from stage 1 is used to subtract the second texture from the result of the previous stage.

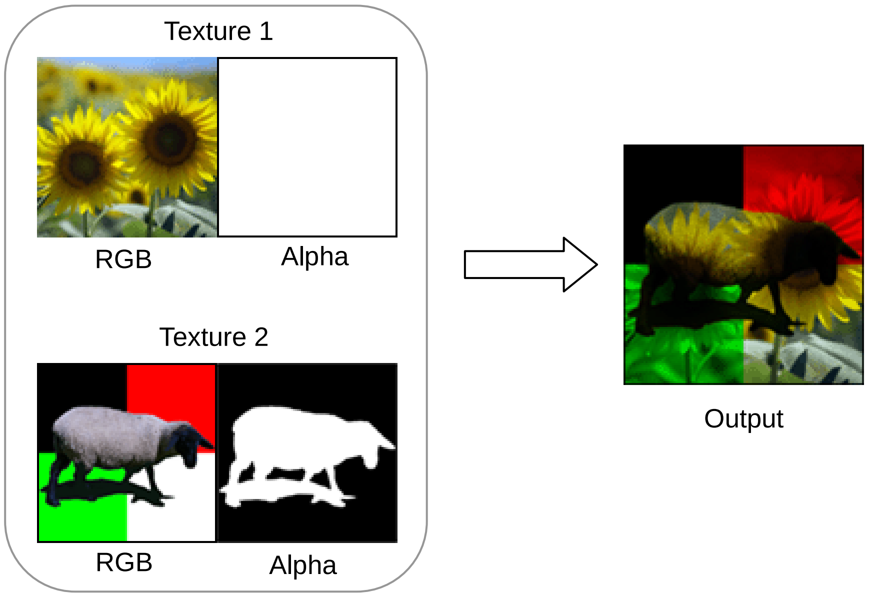
Figure 4-6 Example of Stage Settings for Subtraction



4.2.3 Multiplication

This method multiplies two textures to produce a darker texture, as illustrated in Figure 4-7.

Figure 4-7 Example of Texture Multiplication



4.2.3.1 How Multiplication Works

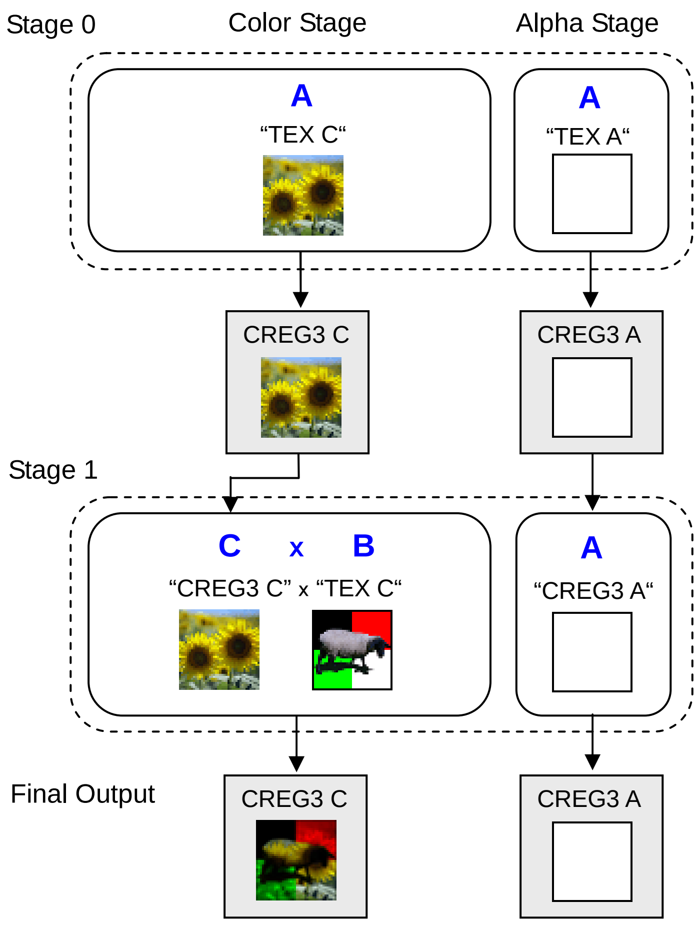
In the TEV computation expression, where 0.0 is specified for **A** and **D**, **C** × **B** is used:

$$\begin{aligned}\text{Result} &= [\,\mathbf{D} + \{(1 - C) \times A + \mathbf{C} \times \mathbf{B}\} + 0.0\,] \times 1.0 \\ &= [\,\mathbf{0.0} + \{(1 - C) \times 0.0 + \mathbf{C} \times \mathbf{B}\} + 0.0\,] \times 1.0 \\ &= \mathbf{C} \times \mathbf{B}.\end{aligned}$$

4.2.3.2 Stage Settings for Multiplication

Multiplication of two textures requires the use of two stages of TEV. In stage 0 (see Figure 4-8), the texture color and alpha are output unmodified to color register 3 by both the color and alpha stages. Next, **C** × **B** from stage 1 is used to multiply the second texture and the result of the previous stage.

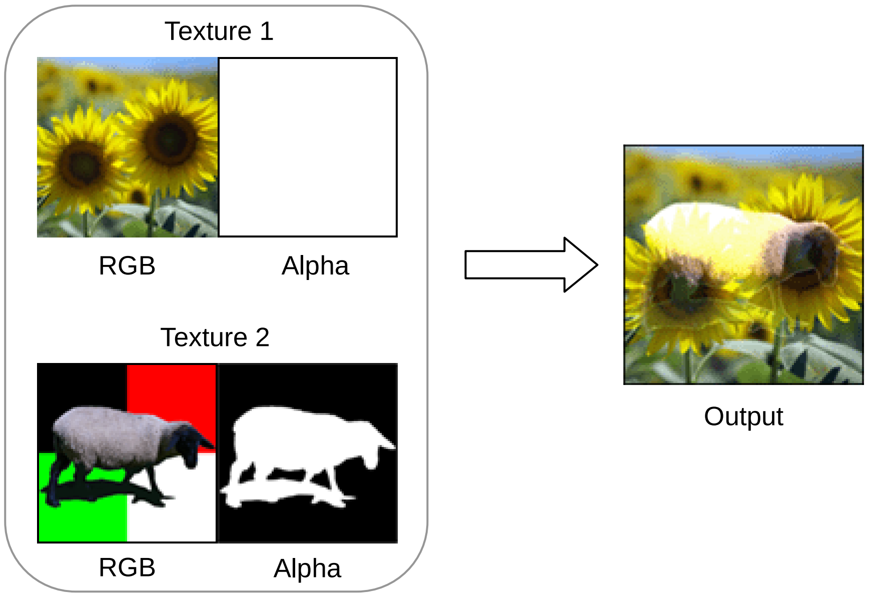
Figure 4-8 Example of Stage Settings for Multiplication



4.2.4 Addition and Multiplication Combined

Combining addition and multiplication allows expression of effects where only the parts that have a semi-translucent or opaque alpha component are added, as illustrated in Figure 4-9.

Figure 4-9 Example of Addition and Multiplication Combined



4.2.4.1 How Addition and Multiplication Combined Works

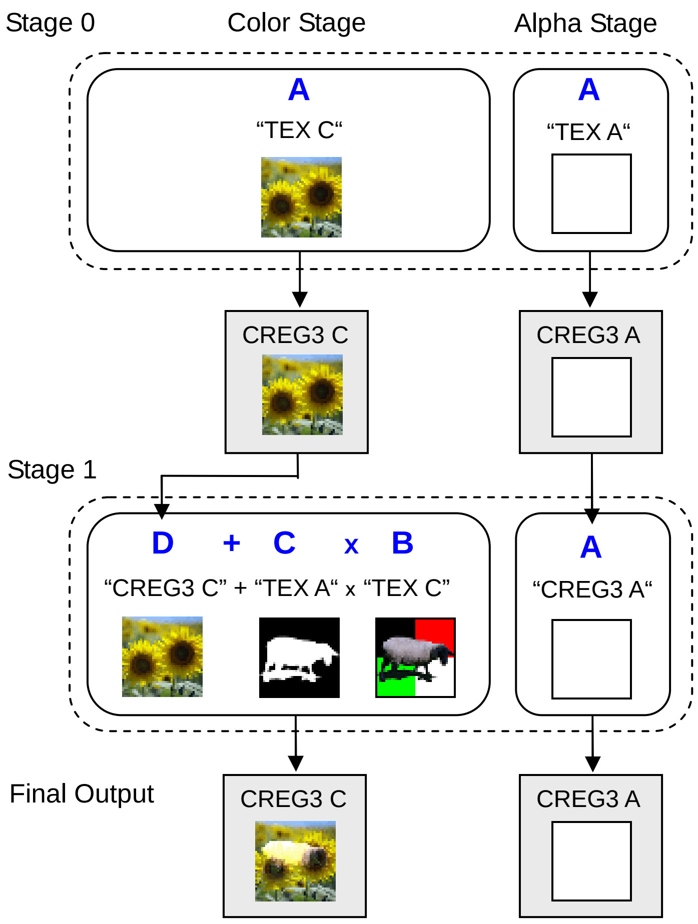
In the example for adding textures, **D + A** was used to add two textures. For addition and multiplication combined, replace **A** with **C** × **B** to multiply the texture color and texture alpha of the second texture, then add the result to that of the previous stage. In other words, only the semi-translucent and opaque portions of the second texture are added. In the TEV computation expression shown directly below, 0.0 is specified for **A**.

$$\begin{aligned}\text{Result} &= [\, \mathbf{D} + \{ (1 - C) \times A + \mathbf{C} \times \mathbf{B} \} + 0.0 \,] \times 1.0 \\ &= [\, \mathbf{D} + \{ (1 - C) \times 0.0 + \mathbf{C} \times \mathbf{B} \} + 0.0 \,] \times 1.0 \\ &= \mathbf{D} + \mathbf{C} \times \mathbf{B}\end{aligned}$$

4.2.4.2 TEV Settings for Addition and Multiplication Combined

The methods of addition and multiplication combined, and addition of textures with alpha components require the use of two stages of TEV. In stage 0 (see Figure 4-10), the texture color and alpha are output unmodified to color register 3 by both the color and alpha stages. Next, **D + C** × **B** from stage 1 is used to add the second texture to the result of the previous stage.

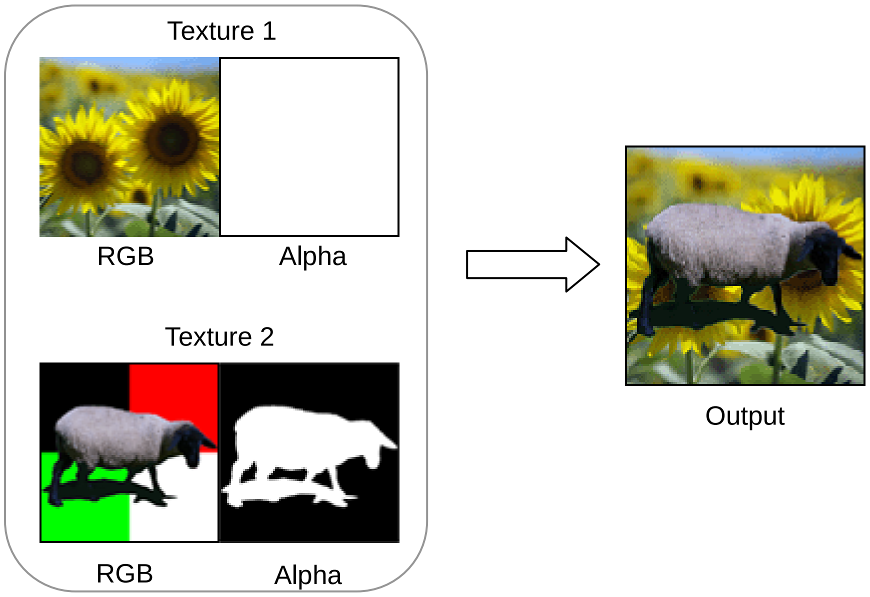
Figure 4-10 Example of Stage Settings for Addition and Multiplication Combined



4.2.5 Decal

This method pastes a texture that has both transparent and opaque texels over another texture. Unlike the addition and multiplication methods (see Figure 4-11), the brightness and coloring of the displayed textures do not change.

Figure 4-11 Example of Decal



4.2.5.1 How Decal Works

In the TEV computation expression, where 0.0 is specified for D, decal uses the **(1 –C)** × **A + C** × **B** part to combine two textures.

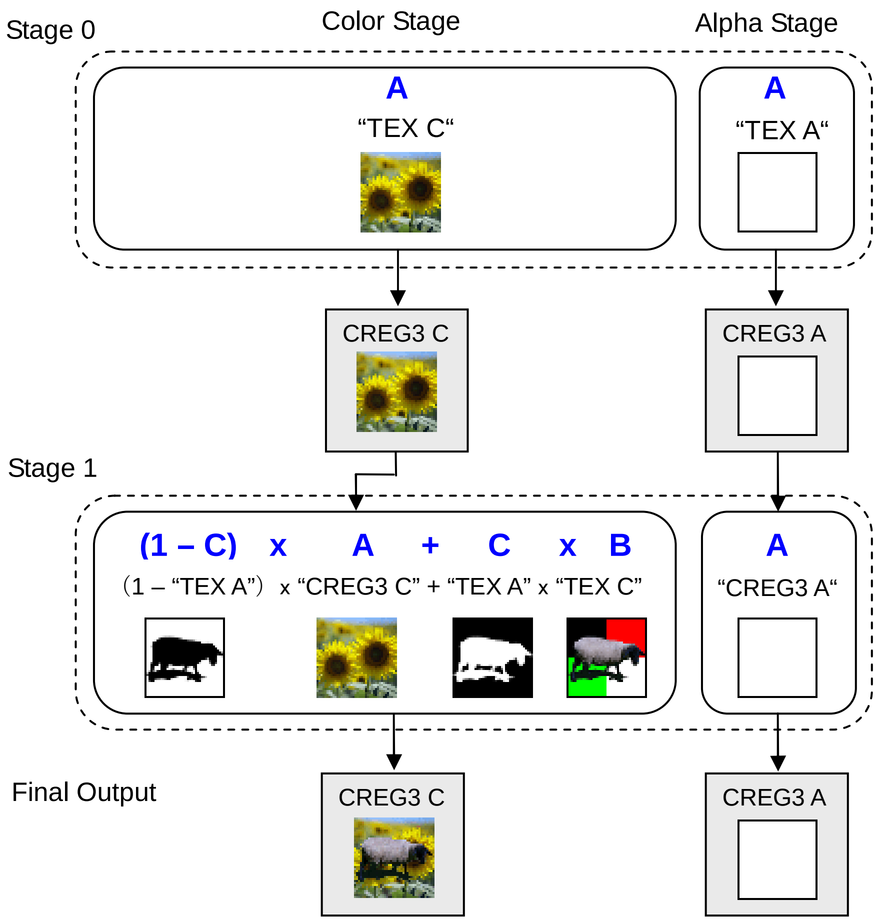
$$\begin{aligned}\text{Result} &= [\,D + \{\,\mathbf{(1-C)} \times \mathbf{A} + \mathbf{C} \times \mathbf{B}\,\} + 0.0\,] \times 1.0\\ &= [\,\mathbf{0.0} + \{\,\mathbf{(1-C)} \times \mathbf{A} + \mathbf{C} \times \mathbf{B}\,\} + 0.0\,] \times 1.0\\ &= \mathbf{(1-C)} \times \mathbf{A} + \mathbf{C} \times \mathbf{B}\end{aligned}$$

Although this expression may look complicated, notice that **(1 –C)** and **C** represent proportions. The larger the value of **C**, the greater the relative weight of **B**. Conversely, the smaller the value of **C**, the greater the relative weight of **A**. When **C** equals 0, the output is **A**. When **C** equals 1, the output is **B**. Decal uses the texture's alpha component for **C**. The effect is similar to masking.

4.2.5.2 Stage Settings for Decal

Using decal to paste one texture onto another requires the use of two stages of the TEV. In stage 0 (see Figure 4-12), the texture color and alpha are output unmodified to color register 3 by both the color and alpha stages. Next, in stage 1, the second texture's alpha component is assigned to **C** and **(1 – C)** × **A** + **C** × **B** is used to combine the two textures. When this is done, the first texture shows where the alpha component is transparent, and the second texture shows where the alpha component is opaque.

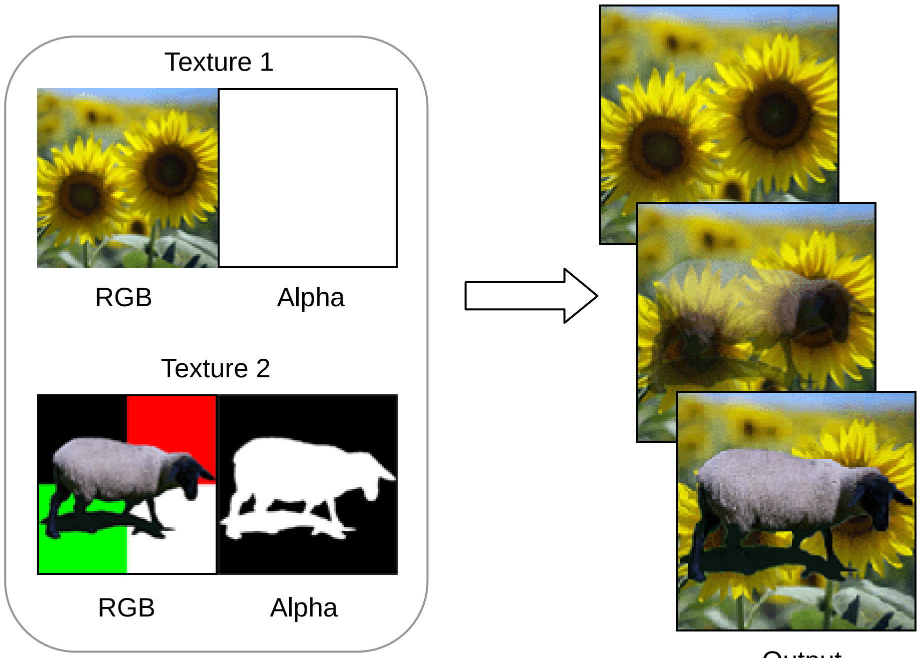
Figure 4-12 Example of Stage Settings for Decal



4.2.6 Decal (Applied)

This method allows applied settings to be configured for the decal method. With these settings (see Figure 4-13), the second texture gradually appears after the first texture is pasted.

Figure 4-13 Example of Decal (Advanced)



Varies depending on the value in the register

4.2.6.1 How Decal (Advanced) Works

As with the standard decal method, in the TEV computation expression (where 0.0 is specified for **D**), the two textures are combined using **(1 – C) × A + C × B**.

$$\begin{aligned}\text{Result} &= [\, D + \{\, \mathbf{(1 - C)} \times \mathbf{A} + \mathbf{C} \times \mathbf{B} \,\} + 0.0 \,] \times 1.0 \\ &= [\, \mathbf{0.0} + \{\, \mathbf{(1 - C)} \times \mathbf{A} + \mathbf{C} \times \mathbf{B} \,\} + 0.0 \,] \times 1.0 \\ &= \mathbf{(1 - C)} \times \mathbf{A} + \mathbf{C} \times \mathbf{B}\end{aligned}$$

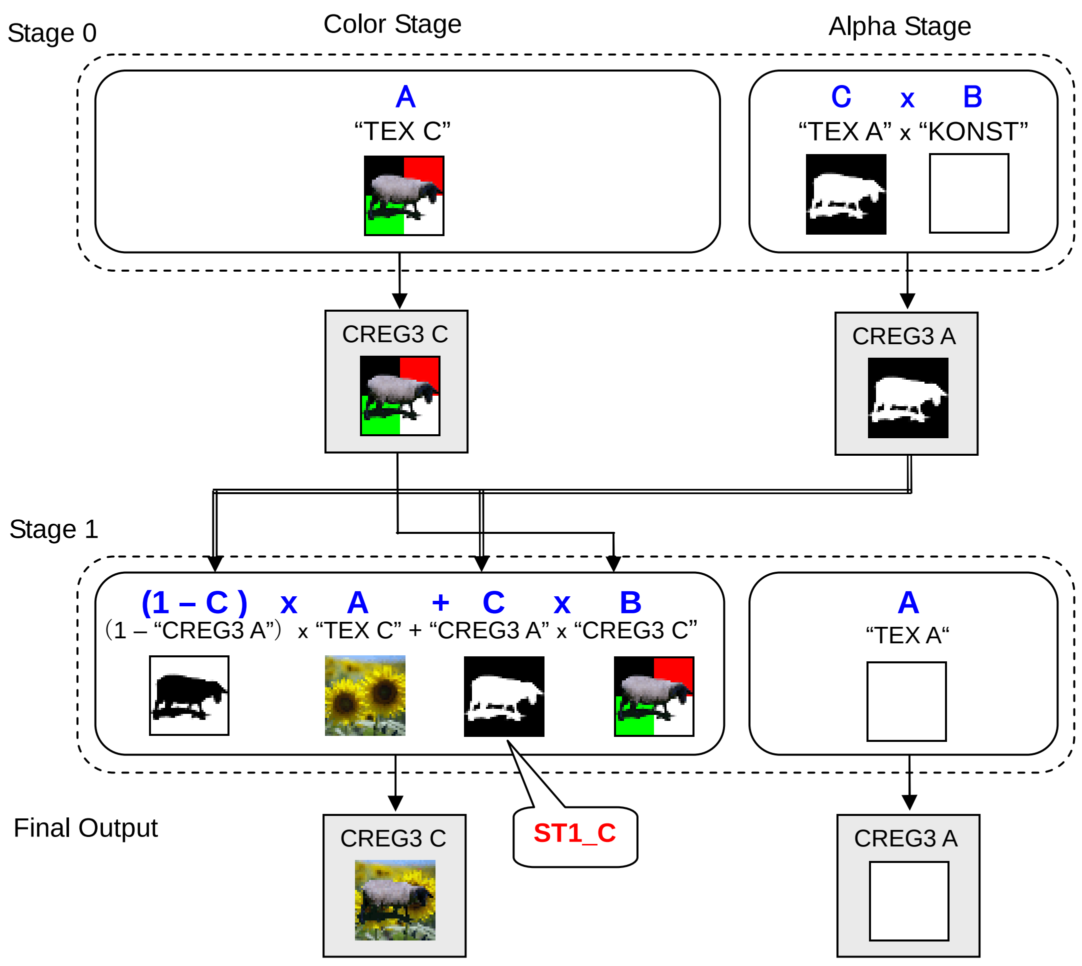
In decal (applied), the order of textures is the reverse of the standard decal's order. The alpha component of the upper-layer texture is multiplied by the value of the constant register. The result is used as the mask.

4.2.6.2 Stage Settings for Decal (Applied)

Using decal (applied) requires the use of two stages of the TEV. In stage 0 (see Figure 4-14), the texture alpha and constant registers are multiplied by the alpha stage and output to color register 3. The texture color is output unmodified to color register 3 by the color stage.

In the color stage of stage 1, the two textures are combined. For the blending ratio, the alpha component from color register 3 of the previous stage is used.

Figure 4-14 Example of Stage Settings for Decal (Applied)



When the value of the constant register is 255, the value for **C** entered in the color stage of stage 1 (henceforth referred to as **ST1_C**) functions like a mask for decal display.

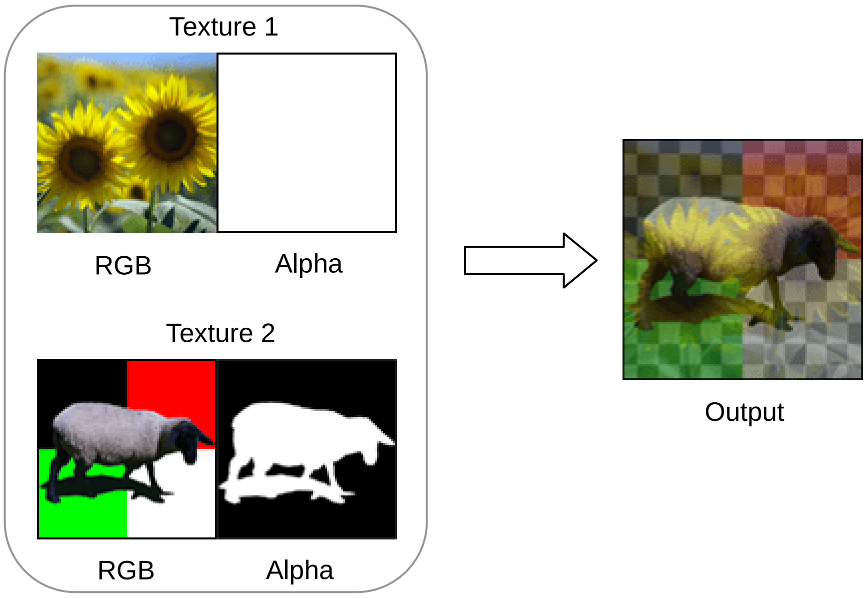
When the value of the constant register is 128, **ST1_C** is half as bright as when the value is 255. Consequently, the first texture is faintly blended.

When the value of the constant register is 0, **ST1_C** is completely black. Only the second texture is displayed.

4.2.7 Proportional Blending

This method blends two textures according to a specified ratio. Although in Figure 4-15 this ratio is 50% – 50%, it can be adjusted to allow for a smooth change of one texture into another. The grid pattern visible in the output indicates the transparency due to alpha.

Figure 4-15 Example of Proportional Blending



4.2.7.1 How Proportional Blending Works

In the TEV computation expression below, where 0.0 is specified for **B** while **C** is used for the blending ratio, the two textures are combined using **D + (1 − C)** × **A**. For **C**, use the components of the color register or the constant register.

$$\begin{aligned}\text{Result} &= [\,\mathbf{D} + \{\,\mathbf{(1 - C)} \times \mathbf{A} + C \times B\,\} + 0.0\,] \times 1.0\\ &= [\,\mathbf{D} + \{\,\mathbf{(1 - C)} \times \mathbf{A} + C \times 0.0\,\} + 0.0\,] \times 1.0\\ &= \mathbf{D + (1 - C)} \times \mathbf{A}\end{aligned}$$

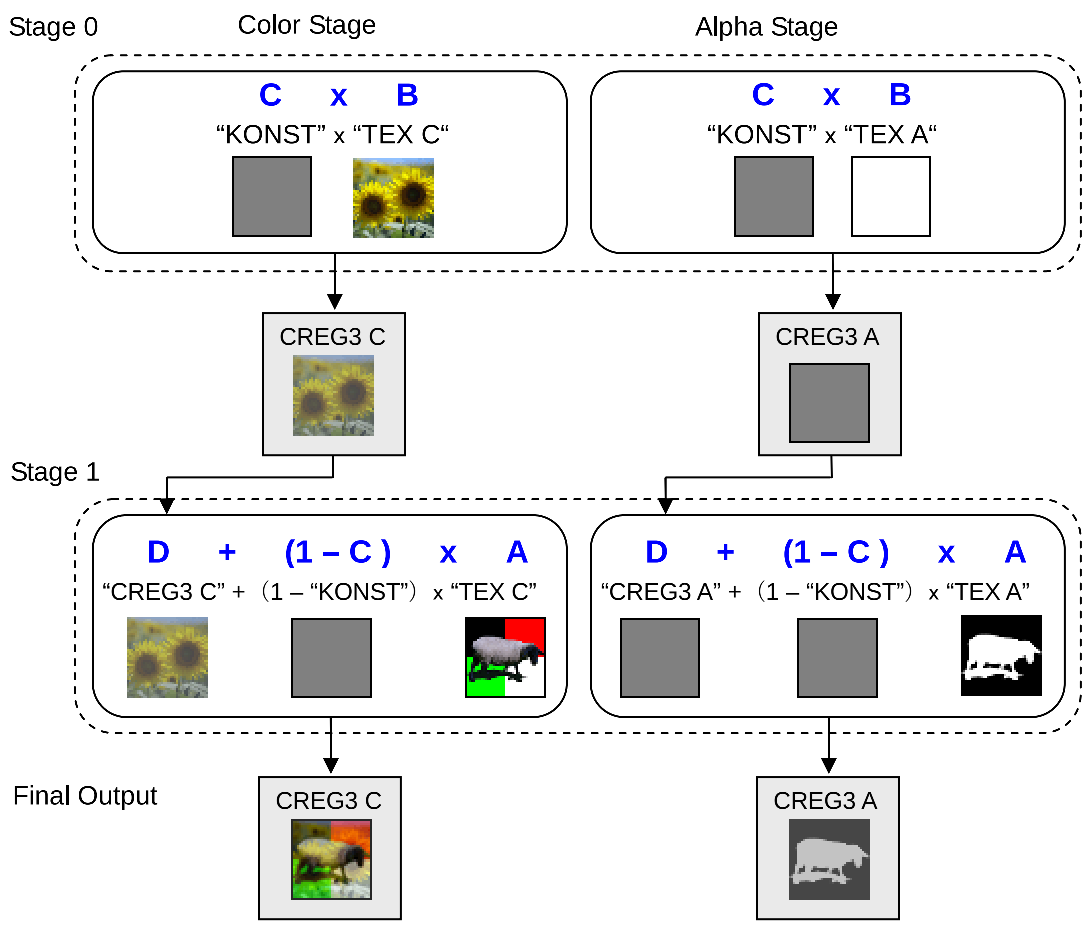
In the previous stage, the first texture was multiplied by the same ratio. The larger the value of **C**, the greater the relative weighting of the first texture. Conversely, the smaller the value of **C**, the greater the relative weighting of the second texture.

4.2.7.2 Stage Settings for Proportional Blending

In Figure 4-16, the **A** component of constant register 3 is used and the value is set to 128. In stage 0, the first texture is multiplied by the ratio set in the constant register in both the color and alpha stages. Next, in stage 1, the second texture is multiplied by the remaining ratio **(1 – C)** and the result is then multiplied by the result of the previous stage.

Although the example in Figure 4-16 uses the **A** component of constant register 3, other registers and components can be used to achieve the same effect. In this figure, "KONST" refers to constant register.

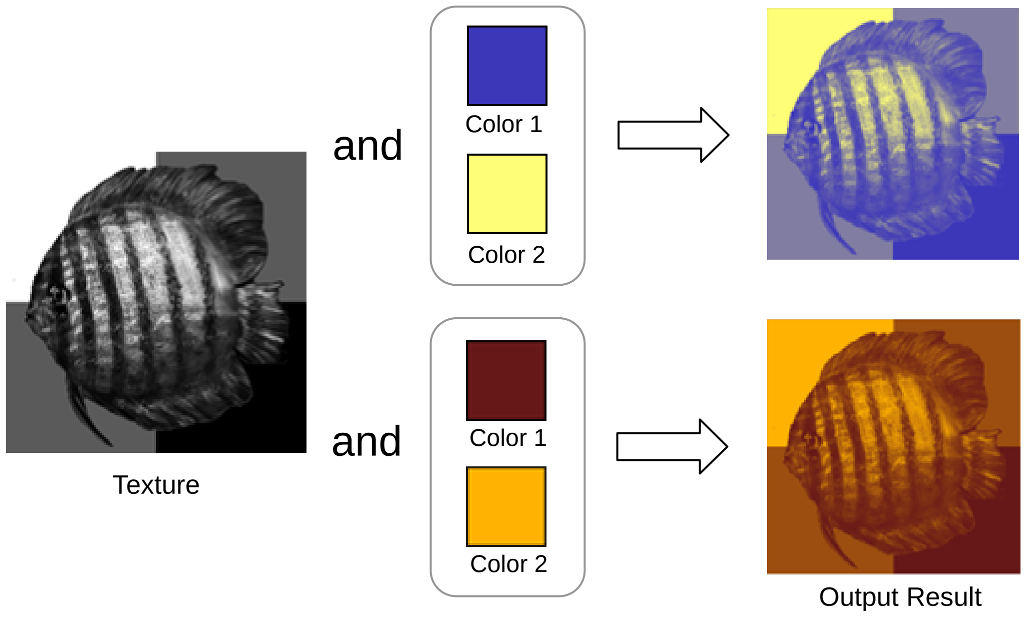
Figure 4-16 Example of Stage Settings for Proportional Blending



4.3 Two-Color Interpolation

Two-color interpolation is one of the basic ways the TEV is used. As shown in Figure 4-17, for grayscale rasterized color and texture color, one color is specified for the bright portions and another for the dark portions. Color is then applied by interpolating these two colors. Two-color interpolation can be used to render sky and ocean scenes.

Figure 4-17 Example of Two-Color Interpolation



4.3.1 How Two-Color Interpolation Works

In the TEV computation expression, where 0.0 is specified for **D**, two-color interpolation uses **(1 – C)** × **A + C** × **B** to combine two colors.

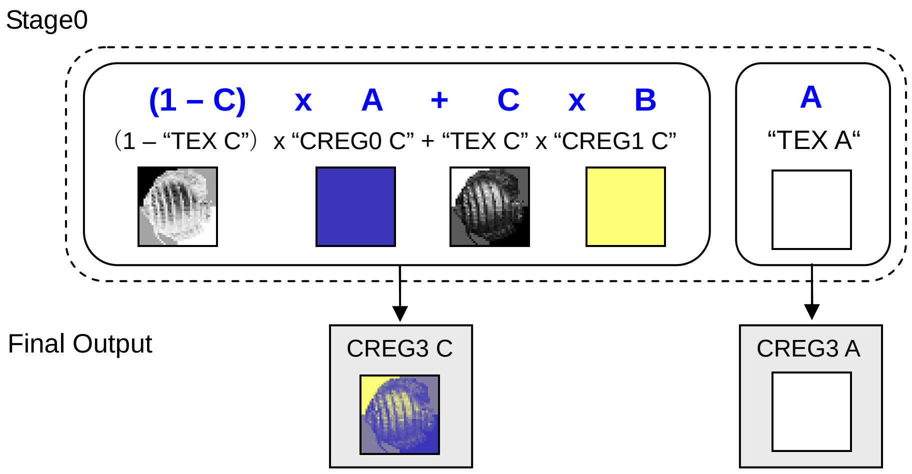
$$\begin{aligned} \text{Result} &= [\,D + \{\,(1 - C) \times A + C \times B\,\} + 0.0\,] \times 1.0 \\ &= [\,0.0 + \{\,(1 - C) \times A + C \times B\,\} + 0.0\,] \times 1.0 \\ &= (1 - C) \times A + C \times B \end{aligned}$$

Normally, a color register is specified for **A** and **B**, and a grayscale texture color or rasterized color is specified for **C**. When this is done, portions of **C** that are lighter (white) are colored with **B**, and portions of **C** that are darker (black) are colored with **A**. As a result, the grayscale shading acquires two colors, **A** and **B**.

4.3.2 Stage Settings for Two-Color Interpolation

Two-color interpolation requires the use of only one stage of the TEV. As illustrated in Figure 4-18, two-color registers (0 and 1) are used to color a grayscale texture. “**1** – **C**” is the same as **C** with the light and dark areas reversed.

Figure 4-18 Example of Stage Settings for Two-Color Interpolation



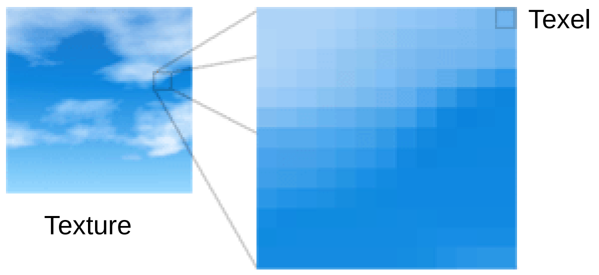
5 Textures

Wii has powerful texture mapping features, including compressed texture, color index texture, texture filter, multi-texture, indirect texture, and mipmap texture. This chapter describes textures.

5.1 Texture Size

While individual picture elements that comprise the frame buffer are called pixels, individual picture elements that comprise the texture image are called *texels*.

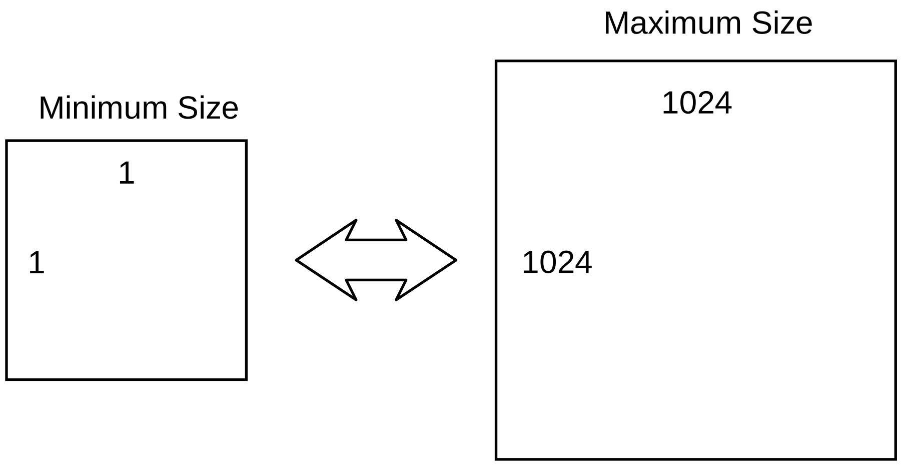
Figure 5-1 Texel



Wii can use textures that range in size from 1 to 1024 texels in width and height (see Figure 5-2). The width and height do not have to be equal, and rectangular textures can also be used.

To use a texture whose resolution exceeds 1024 texels, divide the texture and polygons into a number of parts. To use repeat, mirror, or mipmap, both width and height values must equal a power of 2.

Figure 5-2 Minimum and Maximum Sizes of Textures



5.2 Texture Formats

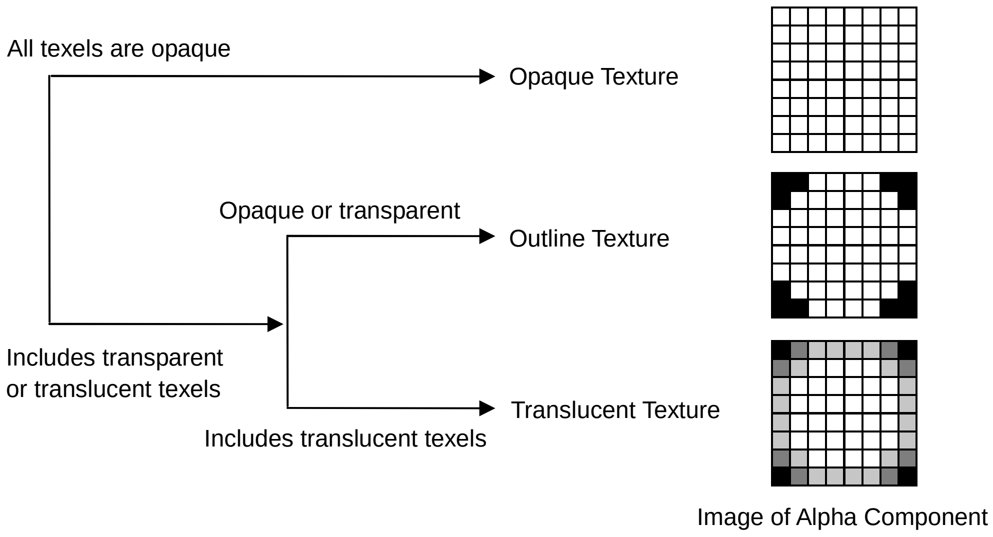
5.2.1 Opaque, Translucent, and Outline

The alpha component of the texture image dictates which texture formats can be used.

In this document, the states of the alpha component are differentiated by using three texture names.

As shown in Figure 5-3, a texture is said to be **opaque** if all texels in the texture image are opaque (α = 255), **outline** if some texels are opaque and others are transparent (α = 0), and **translucent** if any texel is translucent ($1 \leq \alpha \leq 254$).

Figure 5-3 Opaque, Outline, and Translucent Textures



5.2.2 Types of Texture Formats

Table 5-1 shows the eleven texture formats that can be used with Wii.

Table 5-1 Comparison of Texture Formats

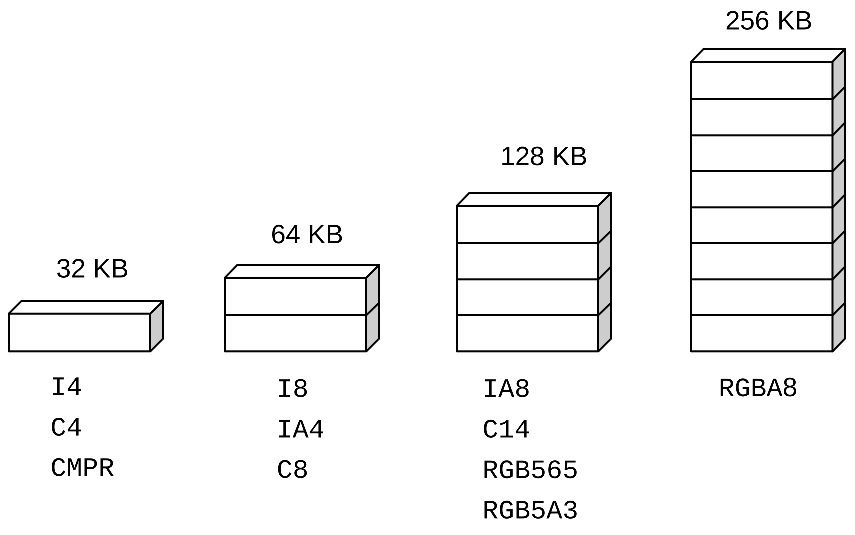
Format (Abbreviation)	Bits Per Texel	Expression of Outlines	Translucence	Tile (see note 1)
Intensity 4-bits (`I4`)	4 bits	Y	Y (16 levels)	8x8
Intensity 8-bits (`I8`)	8 bits	Y	Y (256 levels)	8x4
Intensity+Alpha 8-bits (`IA4`)	8 bits	Y	Y (16 levels)	8x4
Intensity+Alpha 16-bits (`IA8`)	16 bits	Y	Y (256 levels)	4x4
`RGB565`	16 bits	N	N	4x4
`RGB5A3`	16 bits	Y	Y (8 levels)	4x4
`RGBA8`	32 bits	Y	Y (256 levels)	4x4
Compressed Texture (`CMPR`)	4 bits	Y	N	8x8
Color Index 4-bits (`C4`)	4 bits	Y	Δ (Note 2)	8x8
Color Index 8-bits (`C8`)	8 bits	Y	Δ (Note 2)	8x4
Color Index 14-bits (`C14`)	16 bits	Y	Δ (Note 2)	4x4

Note 1: Tiles are explained later in this document. See section 5.2.10 Tiles.

Note 2: For color index, expression of translucence depends on the palettes being used.

The number of bits per texel is different for the different texture formats. If, for example, a 256 x 256 texture is stored in different data formats, the amount of space occupied in memory will differ, as shown in Figure 5-4. Better use of memory can be made by choosing a format to match the texture picture.

Figure 5-4 Comparison of Texture Format and Texture Size (256x256 Texture)

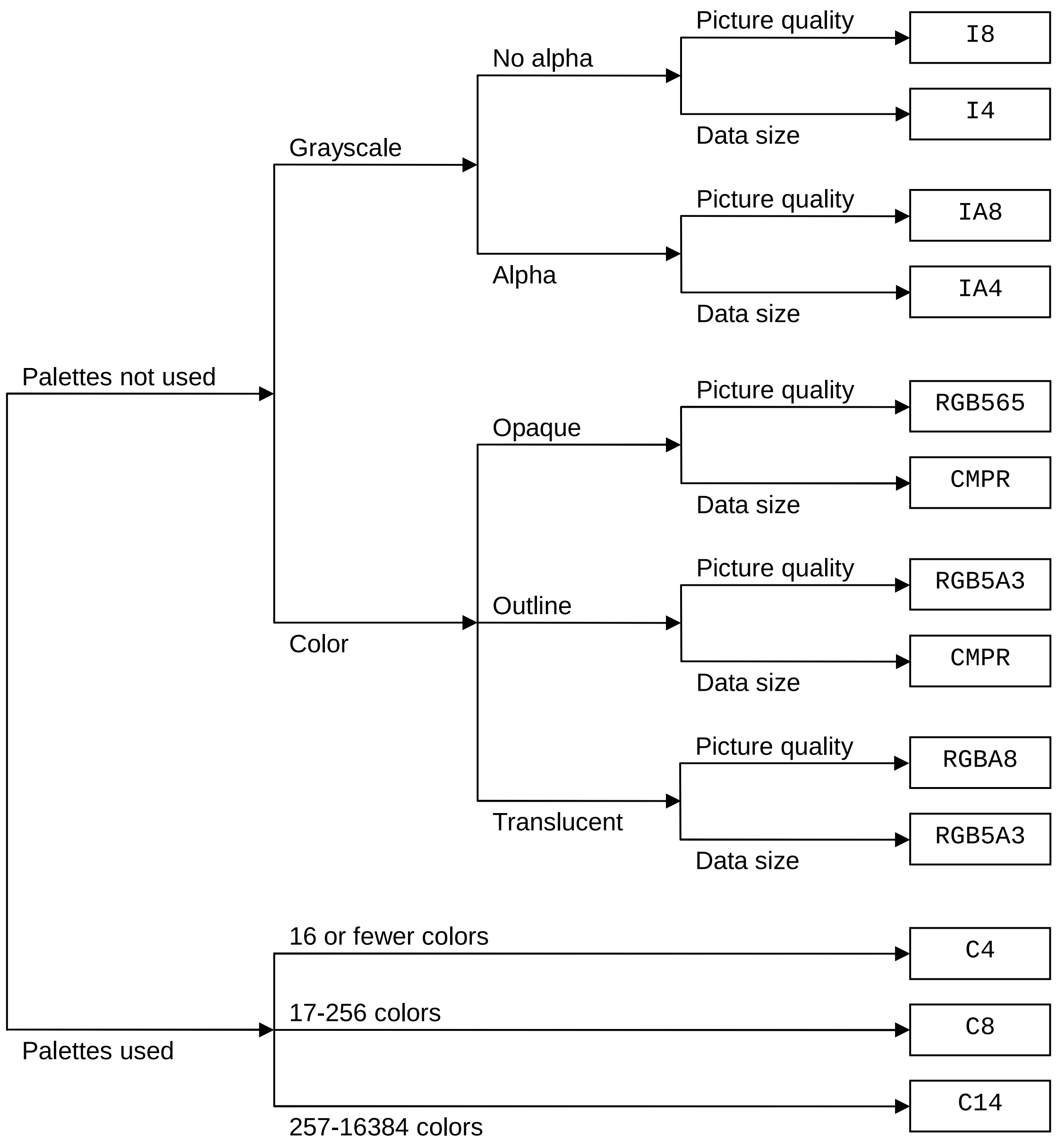


Note: In addition to the data sizes shown in Figure 5-4, separate palette data are required for `C4`, `C8`, and `C14`.

5.2.3 Optimal Texture Format

Figure 5-5 shows how to select the optimal format from the eleven texture formats, depending on the status of RGB components and the alpha component of a texture.

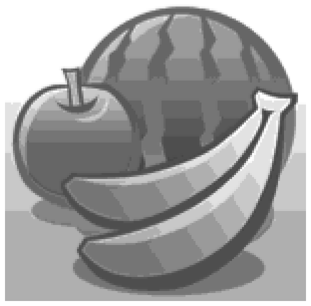
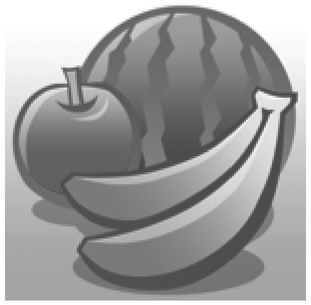
Figure 5-5 Optimal Texture Formats



5.2.4 Intensity

As shown in Table 5-2, `I4` and `I8` are grayscale formats that have only brightness. The texture itself does not have an alpha component, but in the TEV stage settings it can be used as an alpha value.

Table 5-2 Intensity Format Textures

Format	Image	Description
`I4`		Intensity format where each texel is 4 bits. Can be used to express a 16-level grayscale.
`I8`		Intensity format where each texel is 8 bits. Can be used to express a 256-level grayscale.

5.2.5 Intensity Alpha

`IA4` and `IA8` are grayscale formats that have the brightness and alpha components.

Table 5-3 Intensity Alpha Format Textures

Format	Image	Description
`IA4`		Intensity format where each texel is 8 bits. Can be used to express a 16-level grayscale and alpha. **Note:** The grid pattern visible in the picture to the left indicates translucence due to alpha.
`IA8`		Intensity format where each texel is 16 bits. Can be used to express a 256-level grayscale and alpha. **Note:** The grid pattern visible in the picture to the left indicates translucence due to alpha.

5.2.6 RGB

`RGB565` (see Table 5-4) is the format that has RGB components but no alpha component.

Table 5-4 RGB Format Textures

Format	Image	Description
`RGB565`		RGB format where each texel is 16 bits. Includes: • 5-bit red component (32 levels) • 6-bit green component (64 levels) • 5-bit blue component (32 levels)

5.2.7 RGBA

RGB5A and RGBA8 are formats that have RGB components and an alpha component.

Table 5-5 RGBA Format Textures

Format	Image	Description
RGB5A3		In this RGBA format, each texel is 16 bits. RGB5A3 allows the interpretation of individual texels as translucent or opaque to be changed. If the upper bit is 0, the texel is processed as translucent. The red, green, and blue components are 4 bits each (16 levels) and the alpha component is 3 bits (8 levels). If the upper bit is 1, the texel is processed as opaque. The red, green, and blue components are 5 bits each (32 levels). **Note:** The grid pattern visible in the picture to the left indicates translucence due to alpha.
RGBA8		In this RGBA format, each texel is 32 bits. Red, green, blue, and alpha components are 8 bits each (256 levels). RGBA8 gives the most attractive display of pictures that have many colors, but the size of data is also the largest. **Note:** The grid pattern visible in the picture to the left indicates translucence due to alpha.

5.2.8 Compressed Texture

This texture format uses data compression.

Table 5-6 Compressed Texture Format Textures

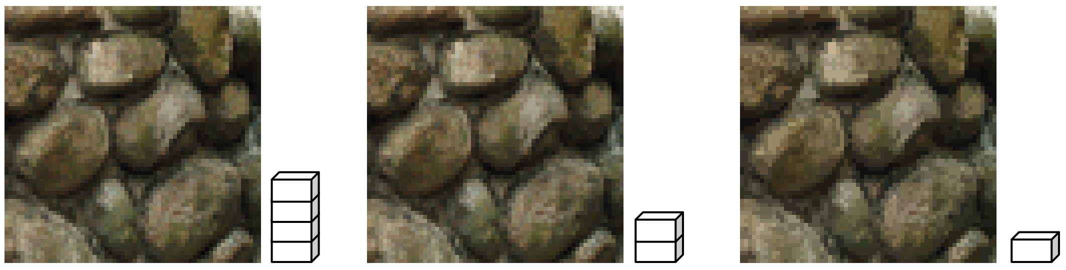
Format	Image	Description
CMPR		Expresses a 4x4 block of 16 texels in 64 bits. Lossy compression is used, so the picture is close to, but not exactly the same as the original. This format cannot express translucence but can express outline (where alpha = 0). Although this is a compression format, it does not place any greater display processing load on the Wii than the other texture formats. **Note:** The grid pattern visible in the picture to the left indicates translucence due to alpha.

Because compressed textures use the lossy compression method, the original picture may not be faithfully reproduced. However, compressed textures reproduce high quality images for natural scenes and images with smooth color gradations.

Generally, this format is optimal for high-resolution images, images that have no translucent texels, and images where the texels have little individual significance.

Figure 5-6 shows the image and data size for a 64 x 64 texture in different texture formats.

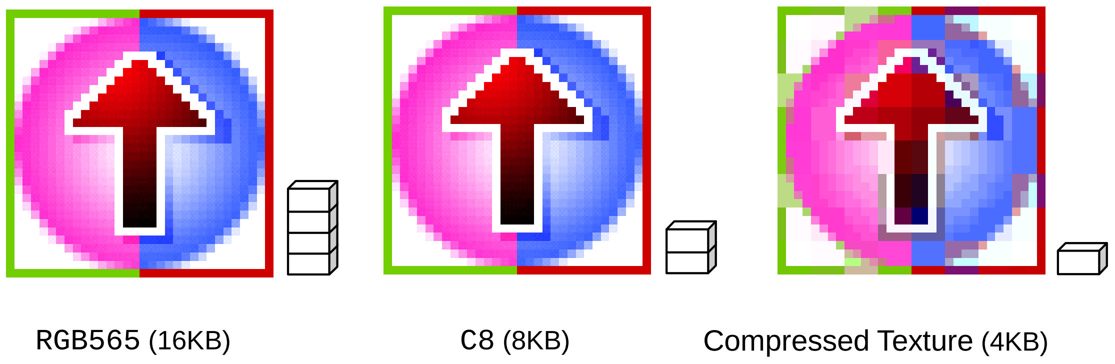
Figure 5-6 Graphic Suited For the Compressed Texture Format



Because loss of detail becomes apparent, compressed textures are not suited for images where neighboring texels often have different colors and the texels have a large individual significance.

Figure 5-7 shows the image and data size for a 32 x 32 texture in different texture formats.

Figure 5-7 Graphic Not Suited For Compressed Texture Format

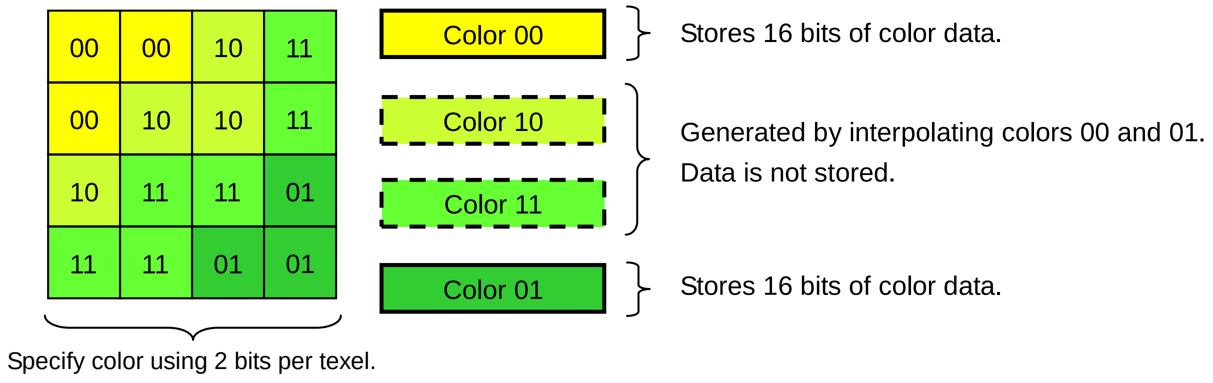


5.2.8.1 How Compressed Texture Works

Compressed texture uses S3TC compression technology.

Each 4 x 4 set of texels is treated as one block. The 4 x 4 set of 16 texels requires at most 16 colors, but this is recreated using 4 colors. The data is stored for only two colors (not four). The other two colors are created using interpolation. The resulting four colors are used to express the 16 texels. Figure 5-8 and Figure 5-9 demonstrate how compressed texture works.

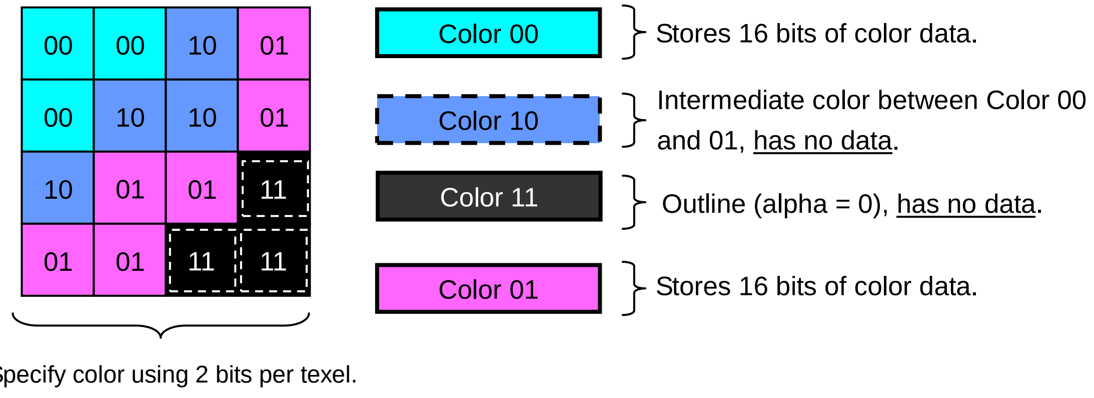
Figure 5-8 Compressed Texture (Generating Two Interpolation Colors)


00
00
10
11
00
10
10
11
10
11
11
01
11
11
01
01
Color 00
Stores 16 bits of color data.
Color 10
Color 11
Generated by interpolating colors 00 and 01. Data is not stored.
Color 01
Stores 16 bits of color data.
Specify color using 2 bits per texel.

Because the color (RGB) data for two colors is stored using 16 bits per color and because each color of the 16 texels is specified using 2 bits per texel, the 16 texels are expressed by 64 bits (16 x 2 + 2 x 16). This corresponds to 4 bits per texel.

The preceding explanation assumes that the two interpolated colors are newly generated from two colors. But they can also be created using an intermediate between the two colors and the outline (alpha = 0), as illustrated in Figure 5-9.

Figure 5-9 Compressed Texture (Using Outline)


00
00
10
01
00
10
10
01
10
01
01
11
01
01
11
11
Specify color using 2 bits per texel.
Color 00
Stores 16 bits of color data.
Color 10
Intermediate color between Color 00 and 01, has no data.
Color 11
Outline (alpha = 0), has no data.
Color 01
Stores 16 bits of color data.

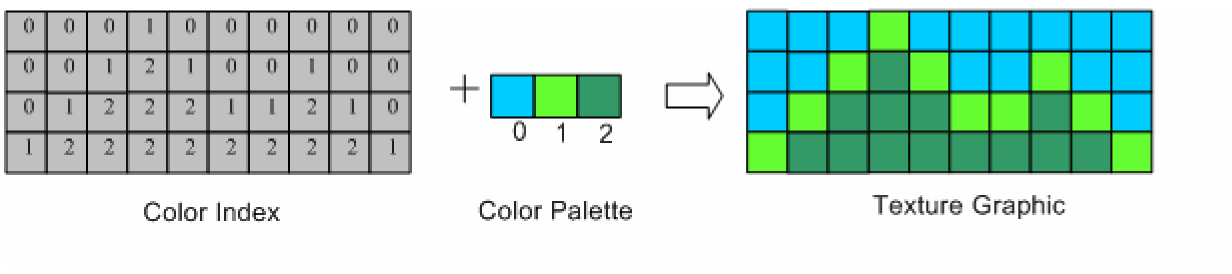
The interpolation method used can be selected in each block and is determined by comparing Color 00 to Color 01 as simple 16-bit values. The interpolation method that does not use outline is selected If Color 00 ≥ Color 01. The interpolation method that uses outline is selected If Color 00 < Color 01.

Because compressed textures use a lossy compression method, the extent of image detail loss is determined when the compressed texture data is generated from the original image. Specifically, it depends on the image type, the two colors stored for each block, and the interpolation method used for each block.

5.2.9 Color Index Texture

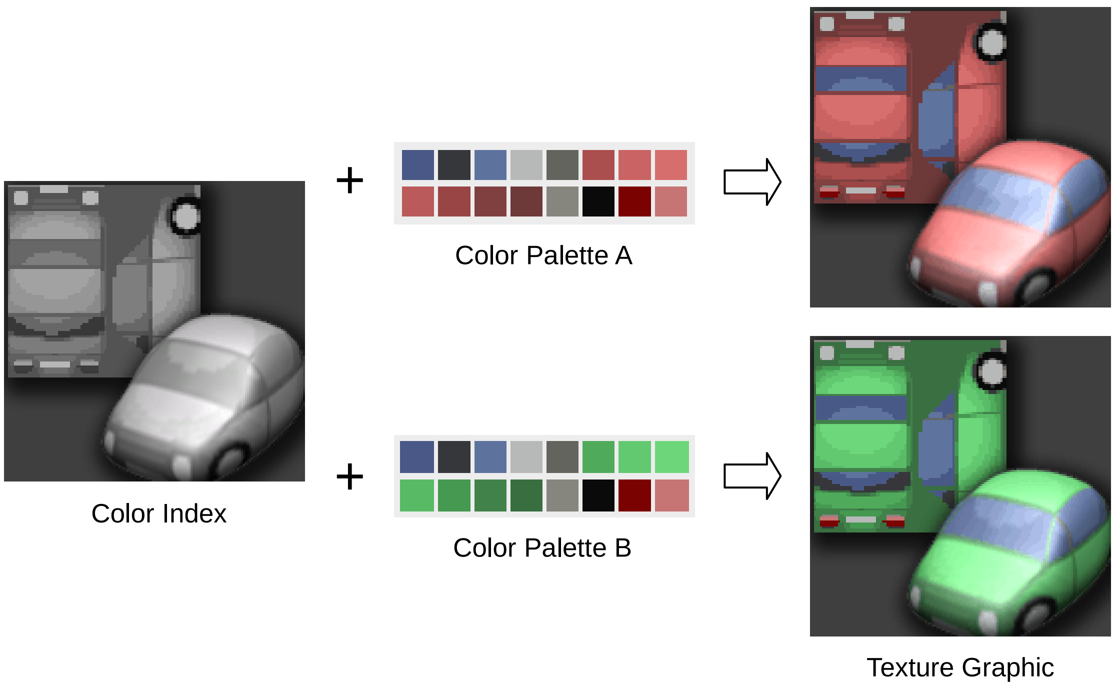
The color index texture formats (`C4`, `C8`, and `C14`) differ from other texture formats that store color information for each texel. These formats store color information in palettes, and each texel stores an index to the color palettes (see Figure 5-10).

Figure 5-10 Color Index



When using color indices as shown in Figure 5-11, not only is data size minimized, but texture colors can also be changed simply by switching color palettes.

Figure 5-11 Switching Color Palettes



5.2.9.1 Color Palettes

Choose one of the following three color palette formats.

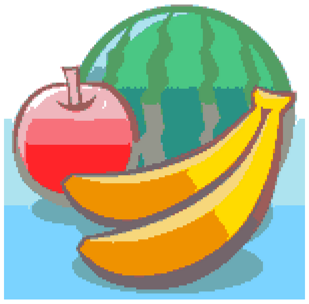
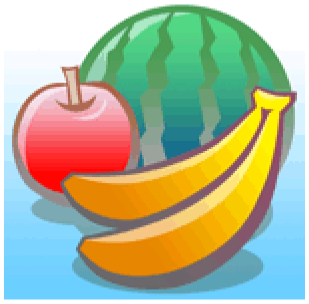
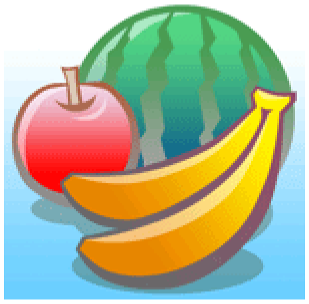
- `IA8`
- `RGB565`
- `RGB5A3`

The color information for each texel is maintained the same way as the aforementioned texture formats. All of these color palette formats maintain one color with 16 bits. There is no 32-bit palette format.

To display a grayscale texture pattern with the best picture quality, use `I8` format. If the pattern is not a grayscale and if it has transparent or translucent alpha values, the `RGB5A3` format is most suitable. If the image is completely opaque, use `RGB565` format.

5.2.9.2 Color Index Format

Table 5-7 Color Index Format Textures

Format	Image	Description
C4		4-bit color index texture. Select colors from a 16-color palette.
C8		8-bit color index texture. Select colors from a 256-color palette.
C14		14-bit color index texture. Select colors from a 16,384-color palette.

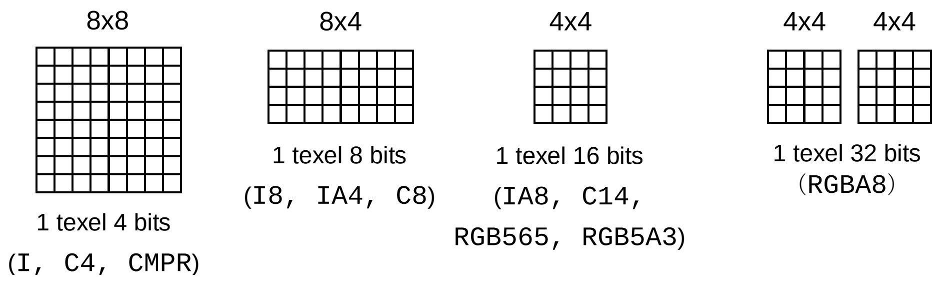
NintendoWare does not support palettes of more than 256 colors.

5.2.10 Tiles

The tile is the smallest image unit for textures that can be used on Wii. To read textures, the GPU uses the tile as its unit, and the width and height of texture data must be in multiples of a tile.

The tile’s size is 32 bytes, same as the size of the Wii's texture cache line. Because the per-texel data size differs for each texture format, the number of texels that constitute a tile also varies by texture format (see Figure 5-12).

Figure 5-12 Texture Format and Tiles



Note: In the 32-bit format, a tile is handled as a set of two 4x4 texels (AR, GB).

For example, if using `I4` format (where one texel is 4 bits), texture data whose width and height are both eight times the size of the texel data must be prepared.

If the texture’s width and height are not multiples of a tile, the texture’s right side and bottom have to be padded with data so that the width and height become multiples of a tile.

If the width and height of a texture are not multiples of a tile, padding is performed automatically by a plug-in so the texture is output as a multiple of a tile.

5.3 Repeat Pasting of a Texture

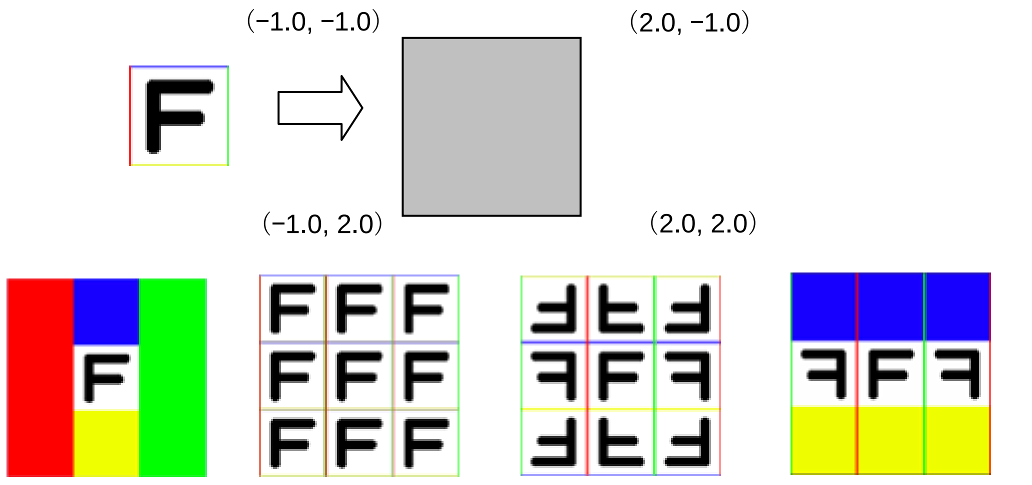
5.3.1 Clamp, Repeat, and Mirror

Wii supports three methods for the repeat pasting of a texture (**clamp**, **repeat** and **mirror**). The three methods can be specified independently in the horizontal and vertical directions.

• Clamp:	Elongates the color at the texture’s edge
• Repeat:	Simply repeats the texture
• Mirror:	Inverts and repeats the texture

Given a quadrangular polygon, for example, the repetitive pasting of a texture once in every direction (left, right, up, and down) would produce the tiling shown in Figure 5-13.

Figure 5-13 Tiling Method



Clamped Vertically and Horizontally
Repeated Vertically and Horizontally
Mirrored Vertically and Horizontally
Mirrored Horizontally, Clamped Vertically

5.3.2 Restrictions for Specifying Repeat and Mirror

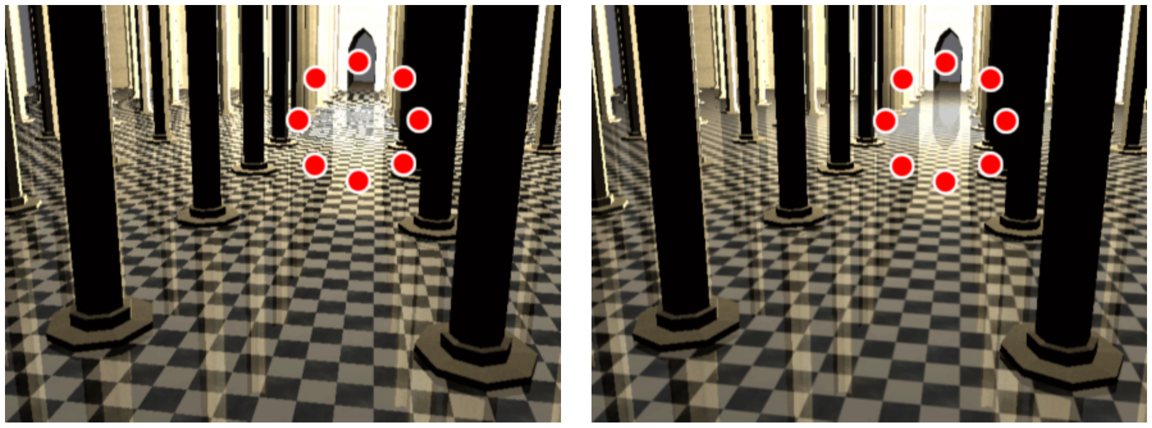
To specify **repeat** or **mirror**, the texel size must be a power of 2 (that is, 4, 8, 16, 32, 64, 128, 256, 512, or 1024), but the width and height do not need to be the same size. Rectangular sizes such as 16 x 8 and 128 x 8 can be used.

5.4 Mipmap

5.4.1 Mipmaps

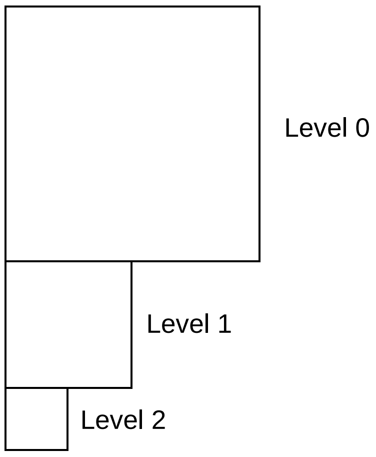
Use mipmaps if the individual texels of a texture mapped to a polygon are smaller than the pixels of the frame buffer (in other words, if the texture is being rendered in a reduced size). As shown in Figure 5-14, mipmapping can reduce moiré patterns. When the polygons are situated far from the camera, mipmapping lightens the load on the GPU.

Figure 5-14 Mipmapping Used to Erase Moiré Patterns



Wii allows up to 11 levels of mipmaps to be used with any of the texture formats. The largest image portion is called level 0 and the surface area of each subsequent level is one-fourth the size of the previous level, as illustrated in Figure 5-15.

Figure 5-15 Illustration of Mipmap Texture



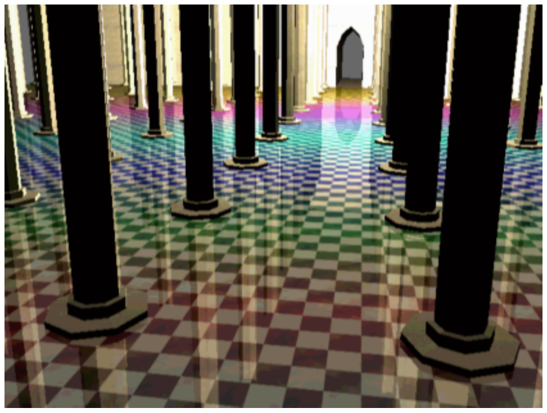
Mipmaps do not require square shapes, but both width and height must be a power of 2 (that is, 4, 8, 16, 32, 64, 128, 256, 512, 1024).

5.4.2 LOD Bias

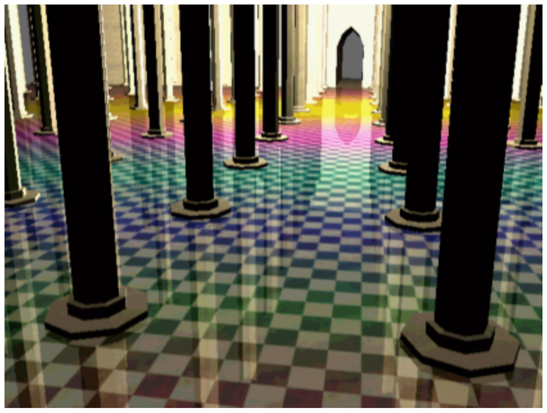
When using mipmaps, textures may become blurry if the texture LOD result is larger than expected and small mipmaps are referenced. To adjust for this, Wii includes the **LOD bias** feature. Adding bias to the LOD computation result alters the way mipmaps are applied. In Figure 5-16, the color is changed at every LOD level, making the change at boundaries easier to recognize.

Figure 5-16 Using LOD Bias to Adjust the Mipmap Level Boundaries

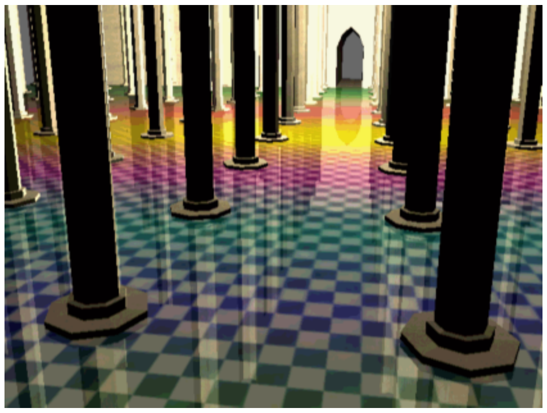
LOD bias = −1



LOD bias = 0



LOD bias = 1

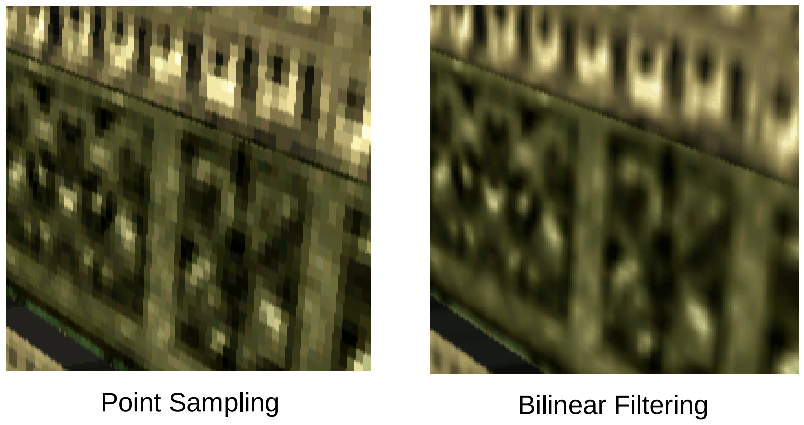


5.5 Texture Filters

Wii's filter feature has two modes for mapping textures (**point sampling** and **bilinear filtering**).

When using point sampling, texels are referenced only when pixel colors are determined. Therefore, the jaggedness at texel boundaries stands out (see Figure 5-17). When using bilinear filtering, interpolation is performed when pixel colors are determined. Therefore, the texel boundaries look smoother.

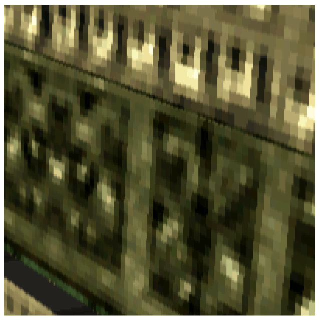
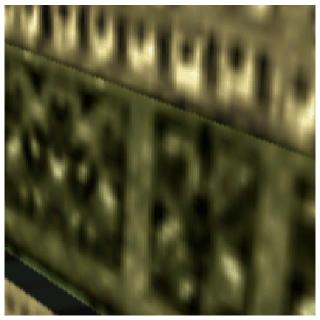
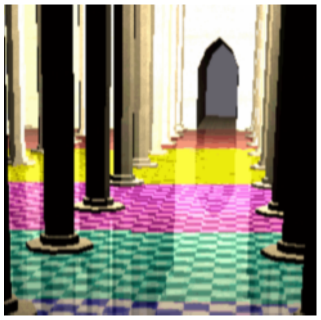
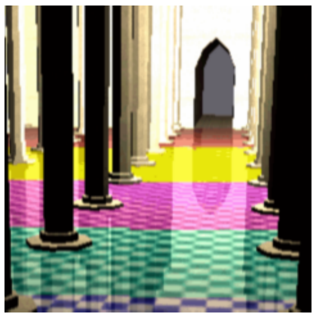
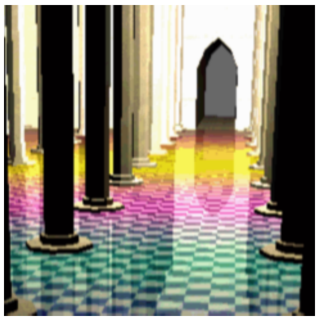
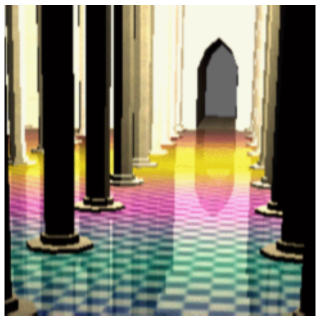
Figure 5-17 Point Sampling and Bilinear Filtering



When texture-mapped polygons are rendered in the frame buffer, individual texels of a texture may be larger than those of the frame buffer. In this case, **the texture is being enlarged** for display. Conversely, when a texture is rendered in the frame buffer in a form smaller than the actual number of its texels, **the texture is being reduced** for display.

For both texture enlargement and reduction, Wii supports separate specification of either point sampling or bilinear filtering as the filter mode. For texture reduction and mipmapping, a texture filter can also be specified for blending the mipmap levels. Table 5-8 illustrates and summarizes the texture filter modes.

Table 5-8 Texture Filter Modes

		Point Sampling	Bilinear Filtering
Mipmap Not Used		Near 	Linear 
Mipmap Used	Sharp shift between levels	Near_Mip_Near 	Linear_Mip_Near 
	Smooth shift between levels	Near_Mip_Linear 	Linear_Mip_Linear 

Enlarging the Texture
- Near
- Linear

Reducing the Texture
- Near
- Linear
- Near_Mip_Near
- Near_Mip_Linear
- Linear_Mip_Near
- Linear_Mip_Linear

Note: Specify a mode for enlarging and a mode for reducing the texture.

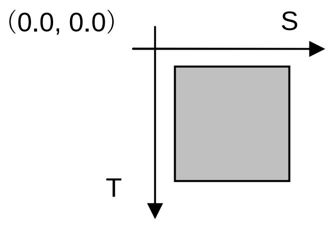
5.6 Texture Mapping

5.6.1 Texture Coordinate Space and Texture Coordinates

When texture coordinates are used to map a texture on Wii, the origin is in the upper-left corner (see Figure 5-18), the forward direction for the S-coordinates is to the right, and for the T-coordinates downward from the origin.

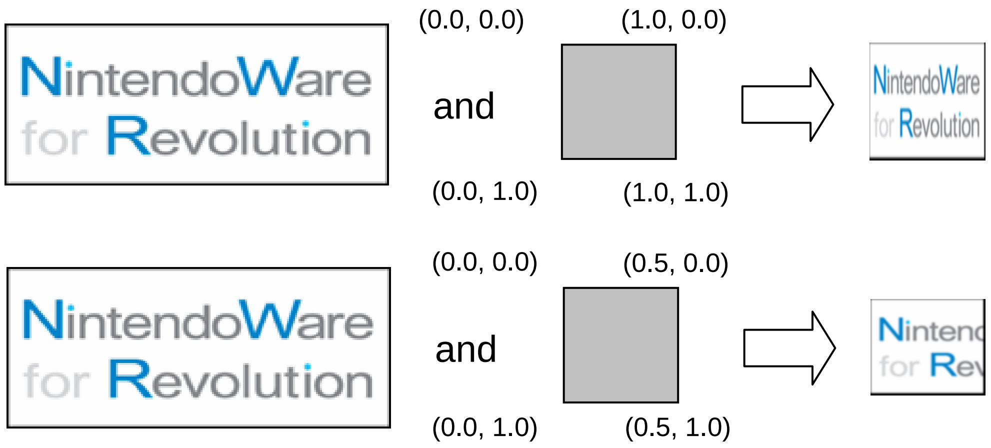
With standard 3DCG tools, the origin (0.0, 0.0) is in the lower-left corner. The forward direction for the U-coordinates is to the right, and for the V-coordinates upward from the origin. Keep in mind that in the texture coordinate space of Wii, the origin position and the upward direction are interpreted differently.

Figure 5-18 Texture Coordinate Space



Regardless of the texture size, mapping fits the texture in the 0.0 – 1.0 range in both the S and T directions. Figure 5-19 shows how the mapping looks when a texture that is long in the horizontal direction is pasted to a square polygon with the ST texture coordinates in the 0.0 – 1.0 range, and when it is pasted to another square polygon with the ST texture coordinates in the 0.0 – 0.5 range.

Figure 5-19 Using Texture Coordinates for Mapping



In the plug-in provided with NintendoWare, the texture coordinates are converted to match the Wii's ST coordinates when the data is output.

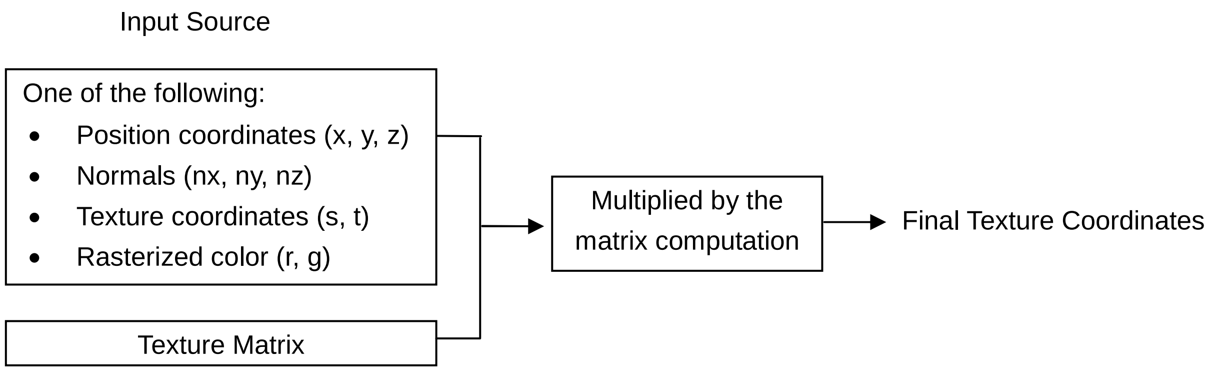
5.6.2 Converting and Mapping Texture Coordinates

5.6.2.1 Converting Texture Coordinates

In addition to using texture coordinates for regular mapping, Wii also performs mapping using vertex position coordinates and normals. This section explains the process of converting texture coordinates.

When texture mapping, vertex position coordinates, normals, texture coordinates, and rasterized colors are used as the input sources for generating the final texture coordinates. As shown in Figure 5-20, the texture coordinates that determine the ultimate position for mapping are determined by multiplying the vertex data (the input source) by the texture coordinate conversion matrix (texture matrix).

Figure 5-20 Converting Texture Coordinates

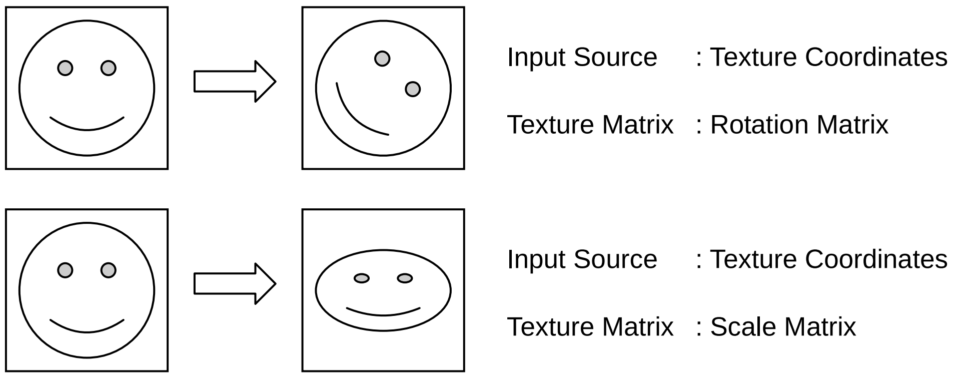

Input Source
One of the following:
• Position coordinates (x, y, z)
• Normals (nx, ny, nz)
• Texture coordinates (s, t)
• Rasterized color (r, g)
Texture Matrix
Multiplied by the matrix computation
Final Texture Coordinates

5.6.2.2 Texture Matrices

A texture matrix of 2 x 4 or 3 x 4 can be used.

If the input source is a set of texture coordinates, and a unit matrix is used as the texture matrix, the texture coordinates themselves will be used for mapping. To set a translation, rotation, and scale matrix for the texture matrix; translate, rotate, and scale the texture (see Figure 5-21). By animating matrix components, the texture's translation, rotation, and scale can also be animated.

Figure 5-21 Using Texture Matrices to Rotate and Scale a Texture

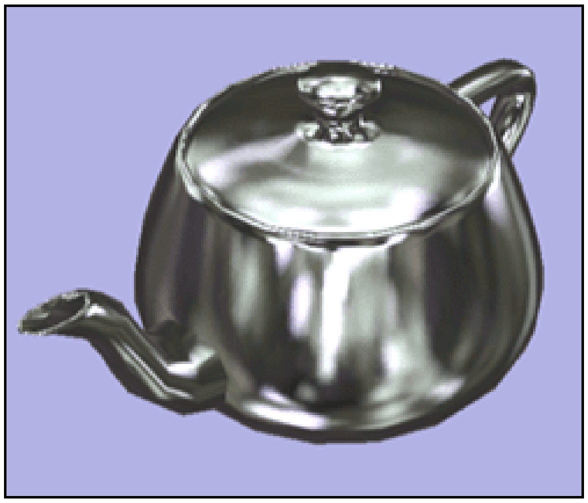

Input Source : Texture Coordinates
Texture Matrix : Rotation Matrix
Input Source : Texture Coordinates
Texture Matrix : Scale Matrix

Note: The center for Rotate and Scale can be adjusted with the Translate component of the texture matrix.

5.6.2.3 Environment Mapping

To make a model look as if the surroundings are being reflected in it, use its normal vectors (nx, ny, nz) for mapping. To depict the texture of metal surfaces, use environment mapping (see Figure 5-22).

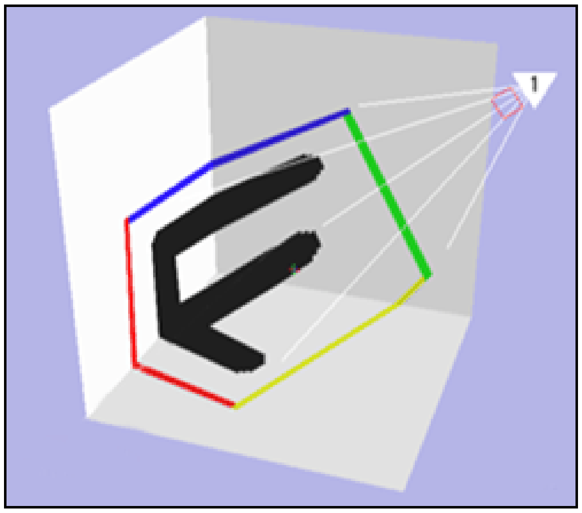
Figure 5-22 Illustration of Environment Mapping



5.6.2.4 Projection Mapping

Use the model's vertex position coordinates (x, y, z) for mapping that looks like a texture is being projected from a specific place. To express filtered sunshine and shadows, use projection mapping.

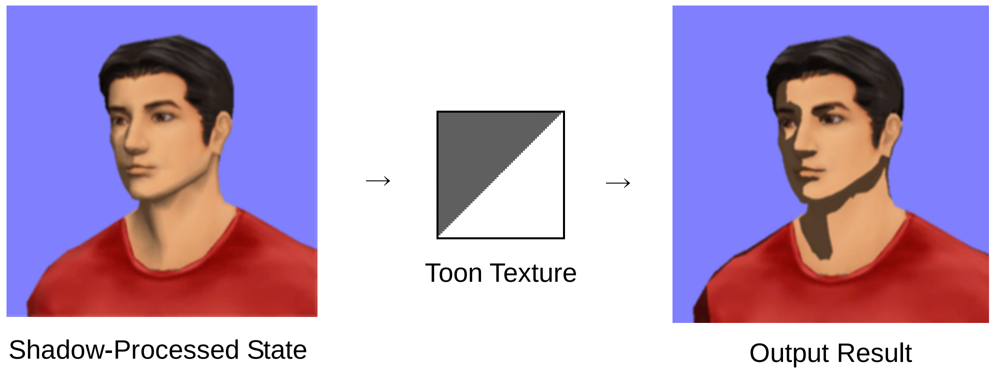
Figure 5-23 Example of Projection Mapping



5.6.2.5 Toon Shading

To depict cell animation, use toon shading.

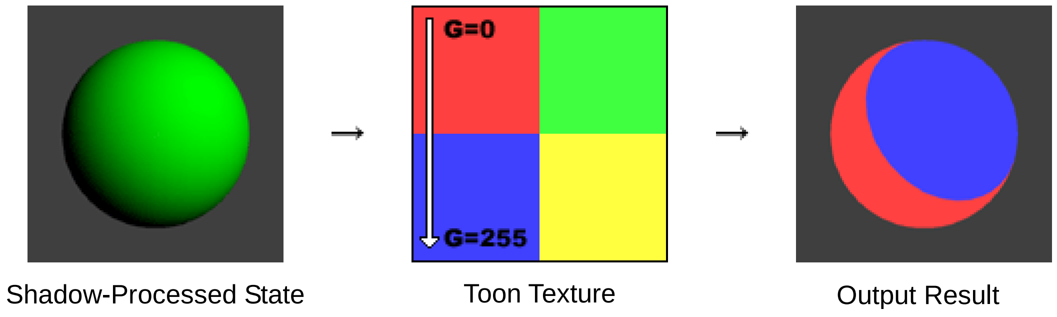
Figure 5-24 Toon Shading



Toon texture is used when the texture coordinates are generated from rasterized color elements. Toon shading refers to the mapping of a texture with the R-component (0 – 255) of the rasterized pixel color for the *s* coordinates (0.0 – 1.0), and the G-component (0 – 255) for the *t* coordinates (0.0 – 1.0).

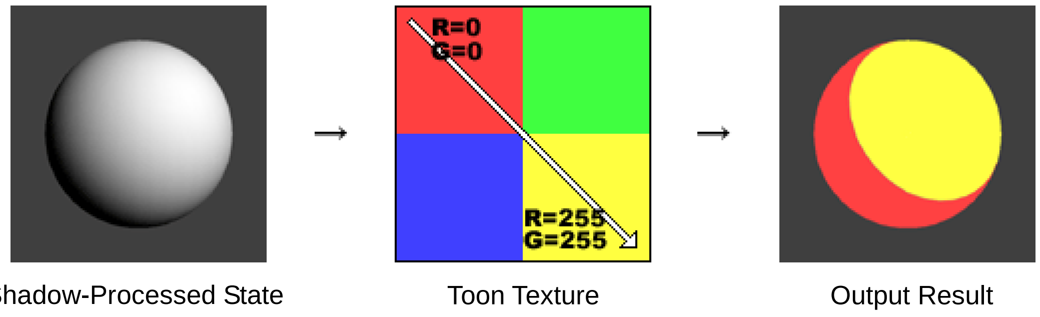
In Figure 5-25, a sphere has shadowing that changes from black (R=0, G=0) to green (R=0, G=255). When the toon texture shown below is used for toon shading, the region between G=0 and G=255 is mapped to the texture from upper-left (s=0.0, t=0.0) to lower-left (s=0.0, t=1.0).

Figure 5-25 Toon Shading, Mechanism 1



If, as shown in Figure 5-26, the sphere is shadowed from black (R=0, G=0) to white (R=255, G=255), the region from (R=0, G=0) to (R=255, G=255) is mapped to the texture from upper-left (s=0.0, t=0.0) to lower-right (s=1.0,t=1.0).

Figure 5-26 Toon Shading, Mechanism 2

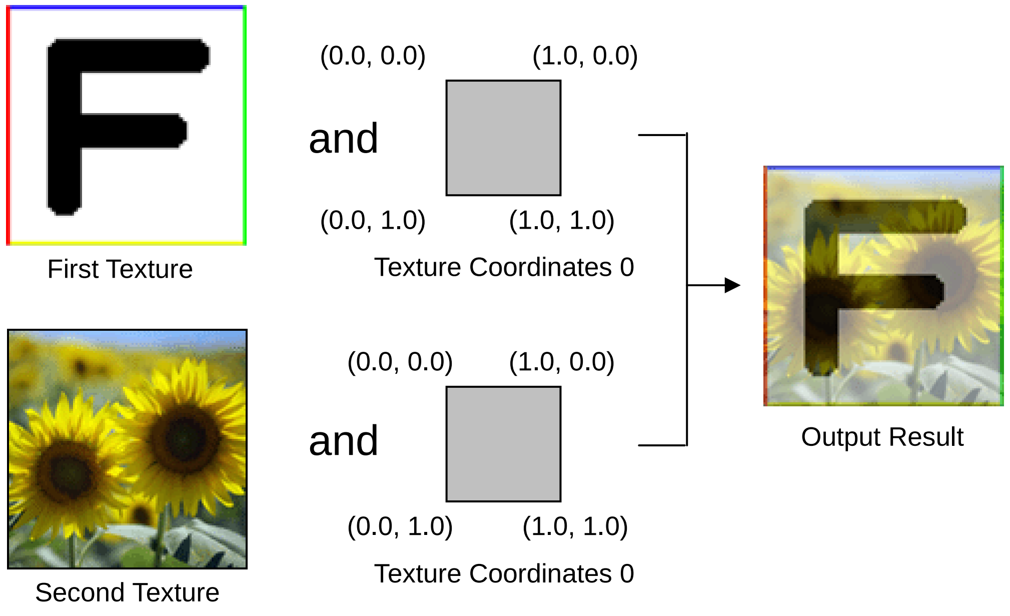


5.7 Mapping Multiple-Layered Textures (Multi-Texturing)

Up to eight textures can be layered on a polygon.

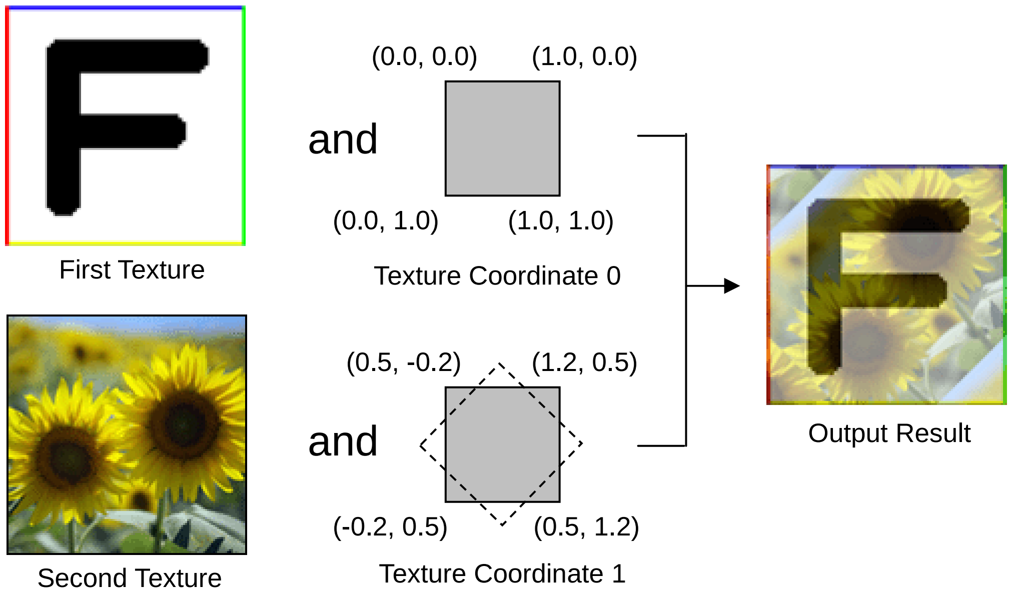
In Figure 5-27, two textures are simply blended. Other methods of layering textures include addition, subtraction, multiplication, and decal. For details, see Chapter 4 Examples of TEV Settings.

Figure 5-27 Mapping of Multiple, Layered Textures



Each vertex can hold up to eight sets of texture coordinates, and different combinations of textures and texture coordinates can be configured.

Figure 5-28 Mapping with Different UV for Each Texture



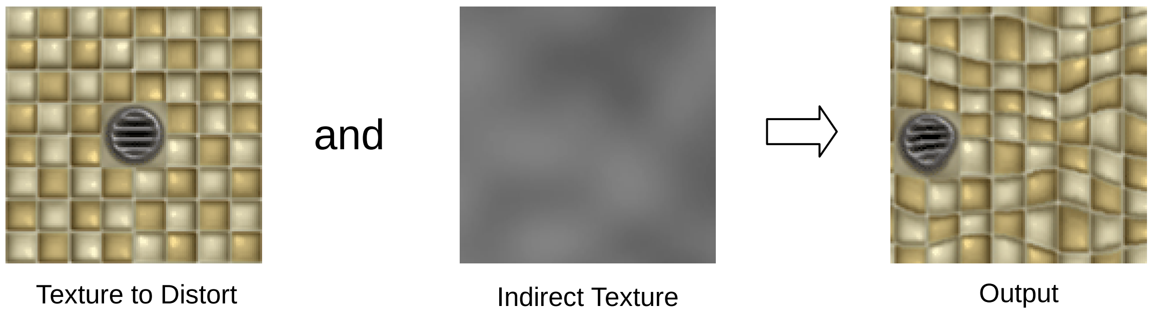
Texture matrices can also be manipulated on each texture and the mapping method can be changed.

5.8 Indirect Textures

5.8.1 What is an Indirect Texture?

To display normal textures that have been distorted, Wii uses the **indirect feature** of the hardware. The texture on which the feature gets applied is the **indirect texture** (see Figure 5-29).

Figure 5-29 Example of Displaying with Indirect Texture



Indirect textures can also be used to depict shimmering water surfaces, mirages, and the effects that are difficult to reproduce with mapping methods and texture matrices.

The indirect texture feature makes use of two textures, but the TEV can process this in one stage.

There are no restrictions on texture format of indirect textures, but the format must have an active alpha component. The standard method is to use `IA8` and to set 256 levels of distortion.

Indirect textures can be mipmapped the same way as normal textures. The use of mipmaps allows the distortion to be varied according to the distance from the camera.

In addition to their use for distorting regular textures, indirect textures can be used for other applications.

When using NintendoWare, indirect textures can be used to express normal vector mapping in object space.

6 Overview of Wii Graphics Hardware

This chapter provides an overview of the main hardware components of the Wii graphics features.

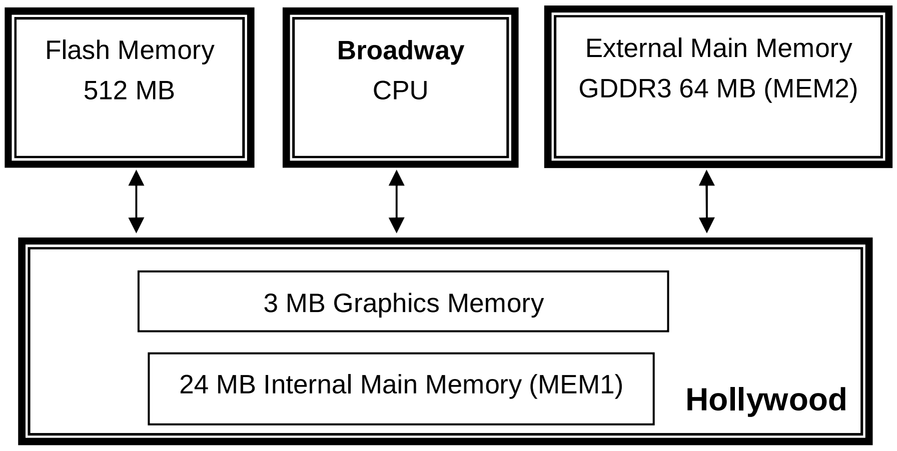
6.1 Hardware Configuration

At the core of Wii hardware is the **Broadway** CPU and the **Hollywood** graphics processor unit (GPU). Broadway CPU performs the processes such as 3D model matrix and animation computations, as well as physical calculations. Hollywood, which primarily renders polygons, comprises a graphics processing unit that includes 3 MB of graphics RAM and 24 MB of internal main memory.

In addition to the Hollywood's 24 MB of internal main memory (MEM1), the main memory also includes GDDR3, which is 64 MB of external main memory (MEM2). In other words, Wii has a total of 88 MB of main memory.

Figure 6-1 contains a schematic of the Wii's graphics-related hardware.

Figure 6-1 Block Diagram of Wii's Graphics Hardware



6.1.1 Broadway

This is the Wii's CPU. Its features include:

- 729MHz operating speed
- 32-bit PowerPC architecture
- Big-endian

6.1.2 Hollywood

This is the Wii's graphics processor. Its main features include:

- Graphics Processing Unit
- 24 MB of internal main memory (MEM1)
    - Ability to hold programs
    - Low latency
- 3 MB of graphics memory

6.1.3 External Main Memory (MEM2)

For external main memory, Wii uses 64 MB of GDDR3 (MEM2). Approximately 10 MB of this memory is used for the System region, so about 54 MB is available to applications. Programs can be placed in MEM2, but access speed is slower than with MEM1.

6.1.4 Graphics Memory

Wii has 3 MB of graphics memory, which it uses for the frame buffer, Z buffer, and texture cache.

6.2 Performance

This section contains information related to performance during rendering.

6.2.1 Processes that Have the Most Impact on Performance

The following processes increase the load on GPU and impact performance during rendering.

- Increasing the number of lights and textures used simultaneously for a polygon
- Increasing the maximum number of TEV stages
- Using indirect textures

6.2.2 Processes that Do Not Impact Performance

The following processes do not impact performance during rendering.

- Using compressed texture and color index affects the load no more or less than other texture formats.
- Changing swap tables does not alter the load.
- Storing vertex data as a fixed point instead of a floating point does not alter the load.
- Using mipmap, except for the RGBA8 format, has no special load associated with it. In fact, due to improved cache efficiency, the load is sometimes lightened when polygons are displayed smaller.