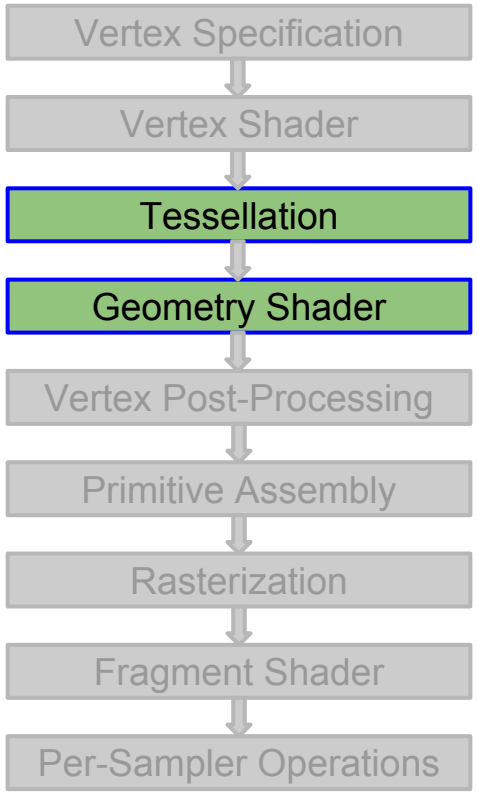
Game Architecture

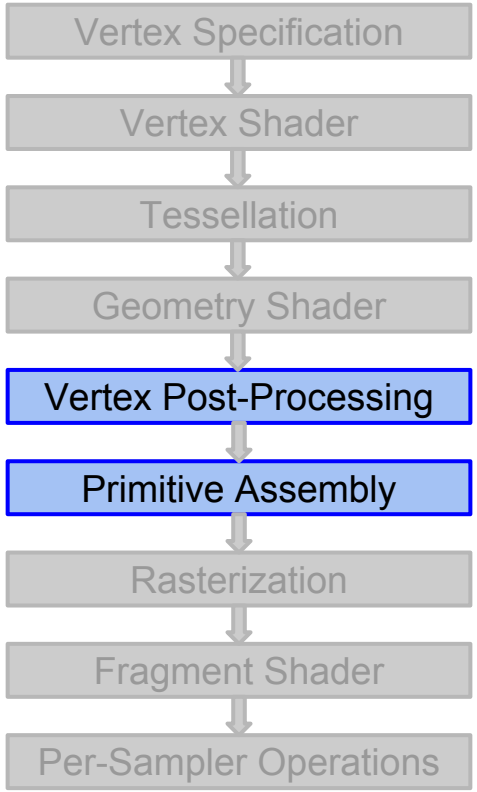
Graphics III

Revenge of the Shader

Programmable Pipeline



Programmable Pipeline



Post-process and Assembly

- Clipping
- Perspective Divide
- Viewport Transform
- Primitive Assembly
- Culling

Post-process and Assembly

- **Clipping**
- Perspective Divide
- Viewport Transform
- Primitive Assembly
- Culling

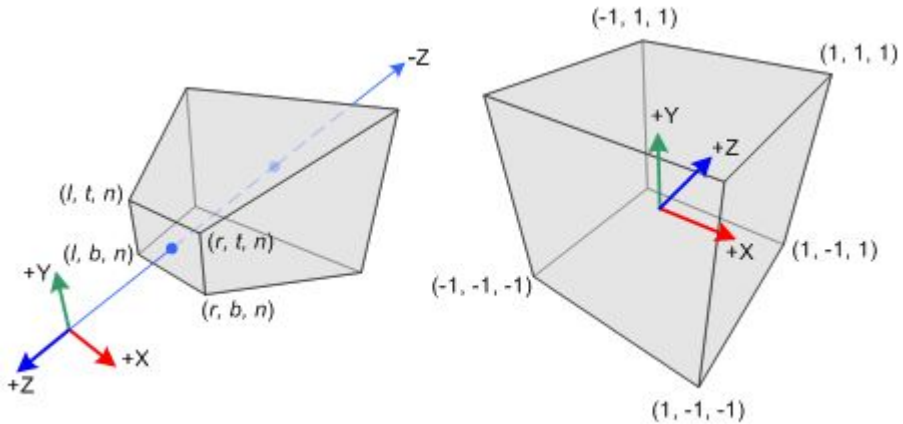


Image courtesy of Khronos Group

Vertex shaders outputs vertex positions in homogenous clip space, which is designed to make clipping as simple as possible.

Portions of primitives outside the view volume are clipped and new primitives are generated.

Primitives which fall completely outside the volume are culled.

Post-process and Assembly

- Clipping
- **Perspective Divide**
- Viewport Transform
- Primitive Assembly
- Culling

This simply divides each clip space coordinate component by the *w* value.

$$\begin{bmatrix} x_{ndc} & y_{ndc} & z_{ndc} \end{bmatrix} = \begin{bmatrix} \frac{x_c}{w_c} & \frac{y_c}{w_c} & \frac{z_c}{w_c} \end{bmatrix}$$

Recall how this *w* value was set up by our projection matrix.

$$\begin{bmatrix} \frac{\frac{dx_v}{a}}{-z_v} & \frac{dy_v}{-z_v} & \frac{z_c}{-z_v} \end{bmatrix}$$

Post-process and Assembly

- Clipping
- Perspective Divide
- **Viewport Transform**
- Primitive Assembly
- Culling

- Transforms the NDC *x* and *y* coordinates into screen space coordinates.
- Viewport is defined by four values:

(*x, y, width, height*)

- The transformation here is simply from range (*-1, 1*) to (x : *width, y : height*)

Post-process and Assembly

- Clipping
- Perspective Divide
- Viewport Transform
- **Primitive Assembly**
- Culling

Literally, gather together sets of vertices that make up each primitive.

- Points
  - One vert per primitive.
- Lines
  - Two verts per primitive.
- Triangles
  - Three verts per primitive.

Post-process and Assembly

- Clipping
- Perspective Divide
- Viewport Transform
- Primitive Assembly
- **Culling**

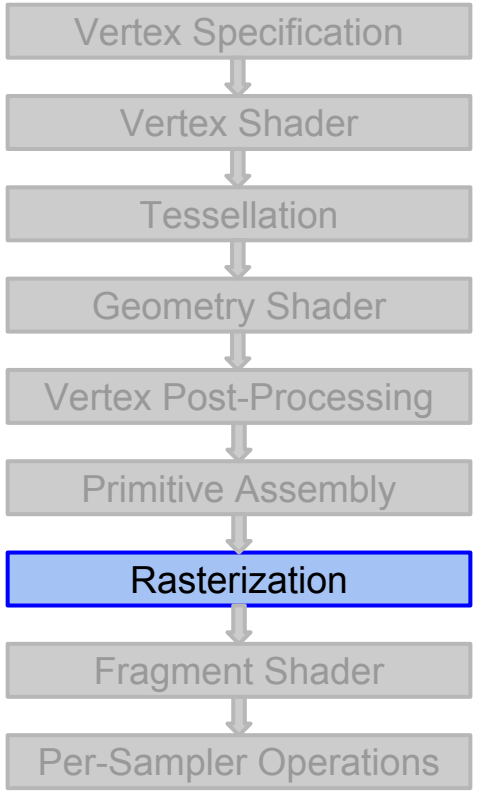
If backface culling is enabled, this discards primitives which are facing away from the camera.

Test by calculating the signed area of the triangle using the cross product.

- Positive - front face.
- Negative - back face.

glCullFace can change the culling mode.

Programmable Pipeline



Rasterization

- Rasterization is the process of breaking a primitive into discrete elements called fragments.
- In short, it can be accomplished using signed distances to lines.
  - Inside a triangle is considered to be the positive side of its edge.
  - Compute signed distance from fragment center to each edge.
  - If all three distances are positive, the fragment is inside the triangle.
- There are faster ways.
  - Tile based rasterization.
  - Hierarchical rasterization.

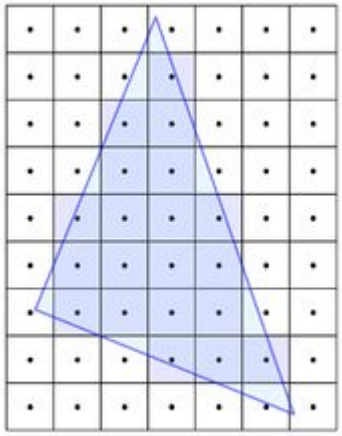


Image courtesy of Wikibooks

Interpolation

- Outputs from the vertex shaders need to be interpolated over the entire primitive surface before being passed to the fragment shader.
  - It turns out for perspective correct result, you need to interpolate values over *w*.
  - For example, *u* / *w* or *v* / *w*.
  - Then multiply the interpolated result by *w*.

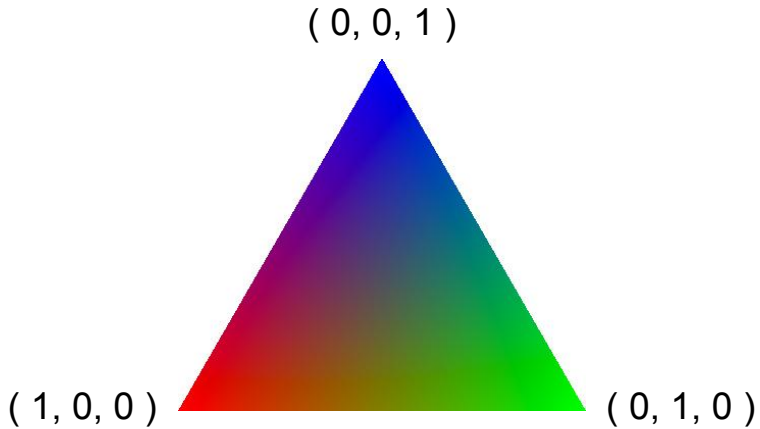


Image courtesy of Alec Jacobson

Barycentric Coordinates

- Coordinates which describe a point within a simplex.
  - Each value in the coordinate can be viewed as a percentage.
  - Multiply/add each vertex by its corresponding coordinate to get interior point.
  - For example, the center of a triangle has barycentric coordinates (0.33, 0.33, 0.33)
  - The centerpoint is the average of all three vertices.
- How you calculate them.

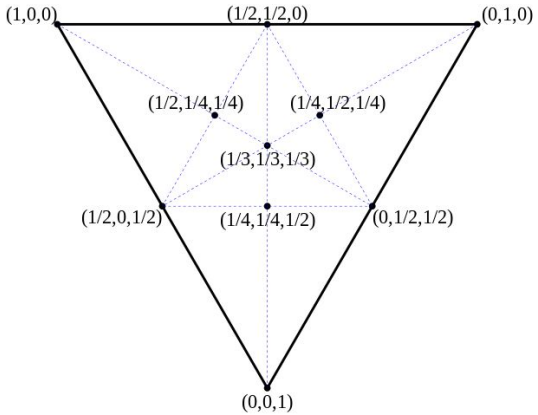
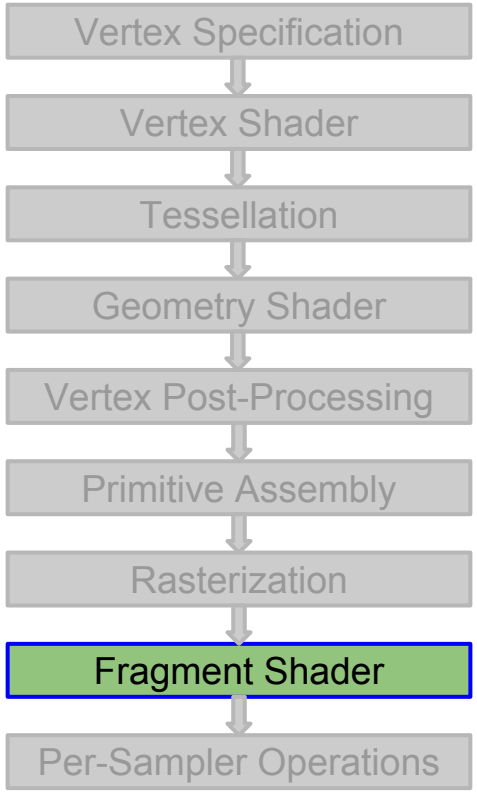


Image courtesy of Wikipedia

Z/Stencil Tests

- Shading multiple fragments for the same pixel is called **overdraw**.
  - More overdraw means worse performance.
- Early Z/Stencil tests can reject fragments from going to the fragment shader.
  - Z test is also known as depth test.
  - There are some conditions which prevent these early tests:
    - Alpha blending
    - Pixel shaders which reject based on depth
    - Pixel shaders which write custom depth values
- Late Z/Stencil tests are the original implementation.
  - These tests occur after shading, so we may throw out completed work.

Programmable Pipeline



Fragment Shaders

- Modern fragment shaders perform:
  - Lighting
  - Complex post-processing
    - SSAO
    - Blur
    - Depth-of-Field
    - Bloom
    - Anti-aliasing
  - Texture sampling
  - Normal distortions
  - Color grading and tonemapping
  - Downsampling
- Speaking of textures...

Textures

- Textures are nothing more than additional buffers of data.
- How they’re interpreted makes them textures.
  - The API may store them in different memory because the data is ‘texture.’
  - The API may also only allow texture data to be passed to certain functions.
- A **texel** is the fundamental unit of a texture.
  - A texel to a texture is a pixel to a display.
- **Sampling** is the process of accessing texel data by providing sample **coordinates.**

Texture Addressing

- Texture coordinates are specified on range [0,1].
  - (0,0) is the bottom left, while (1,1) is the top right.
  - Though this depends on your API.
- However, coordinates are allowed to be given outside the range.
- **Addressing modes** indicate how to deal with out-of-range coordinates.

Texture Addressing Modes

- Wrap
- Mirror
- Clamp
- Border color

Texture Addressing Modes

- **Wrap**
- Mirror
- Clamp
- Border color

Image courtesy of MSDN

Texture Addressing Modes

- Wrap
- **Mirror**
- Clamp
- Border color

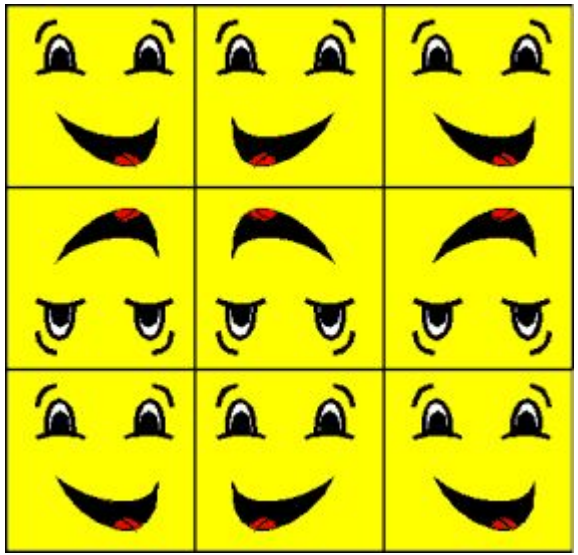


Image courtesy of MSDN

Texture Addressing Modes

- Wrap
- Mirror
- **Clamp**
- Border color

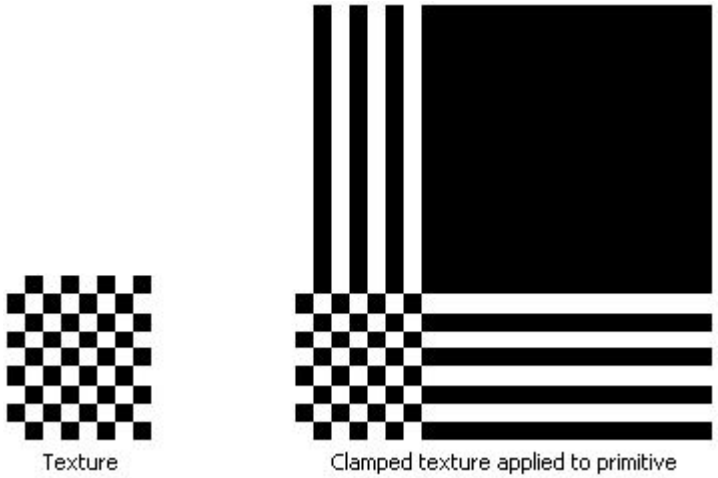


Image courtesy of MSDN

Texture Addressing Modes

- Wrap
- Mirror
- Clamp
- **Border color**

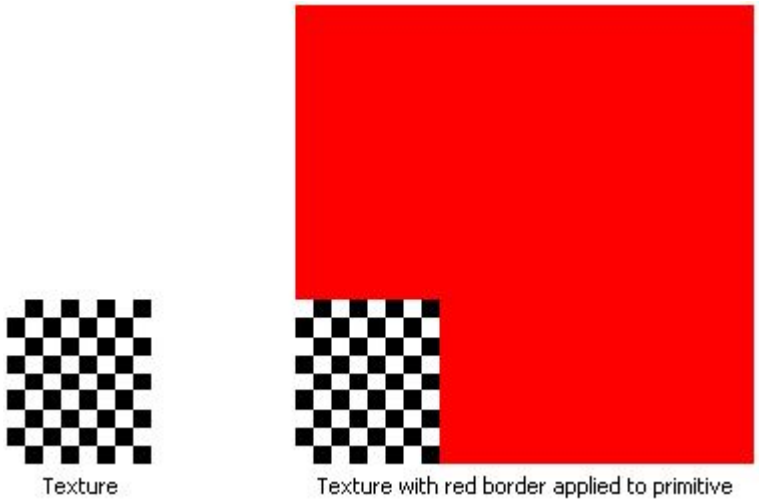
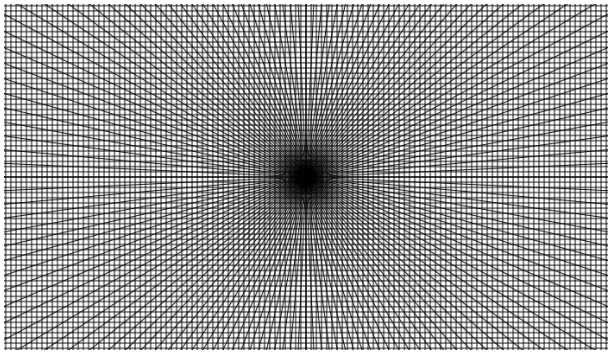


Image courtesy of MSDN

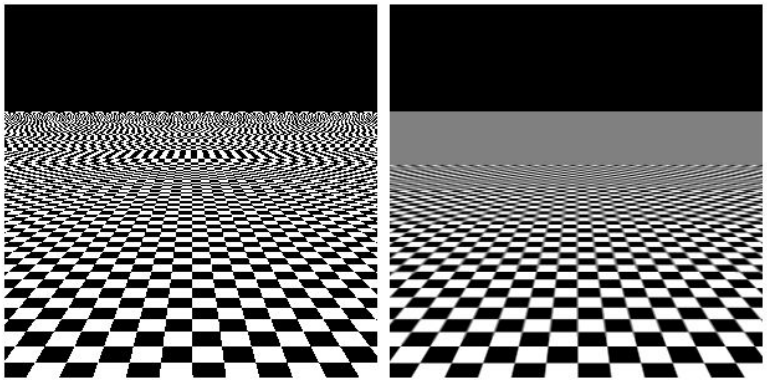
Mipmapping

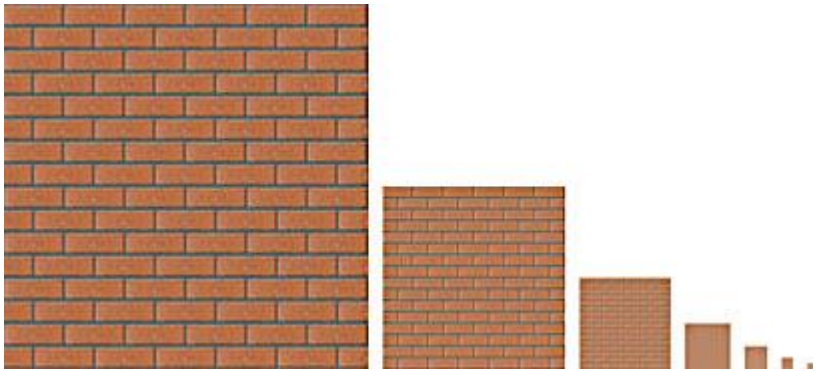
- **Texel density** describes a ratio of texels to pixels.
  - A full-screen primitive with a texture resolution matching the display resolution has a texel density of 1.
  - If the same primitive is viewed from a distance, there are a greater number of texels viewed through a single pixel.
- Artifacts appear the further texel density is from 1.
  - Texel density less than 1 makes the texture look low-resolution.
  - Texel density greater than 1 can lead to moire banding patterns.



Mipmapping

- **Mipmaps** attempt to maintain a texel density close to 1.
- Sequences of lower resolution textures.
  - Each successive texture is half the resolution of the previous.
  - For example, 64x64 | 32x32 | 16x16 | 8x8 | 4x4 | 2x2 | 1x1





Images courtesy of maxogden via Github Gist and Flylib

Filtering

- Nearest
- Bilinear
- Trilinear
- Anisotropic

Texture coordinates for a fragment rarely fall at the exact center of a texel.

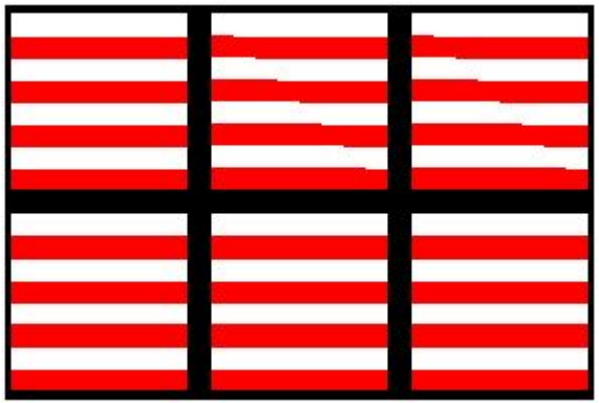
Filtering is the method by which surrounding texels are blended to create the final color for the texture sample.

Filtering

- **Nearest**
- Bilinear
- Trilinear
- Anisotropic

Obtain the texel whose center is closest to the sample location.

- GL_NEAREST



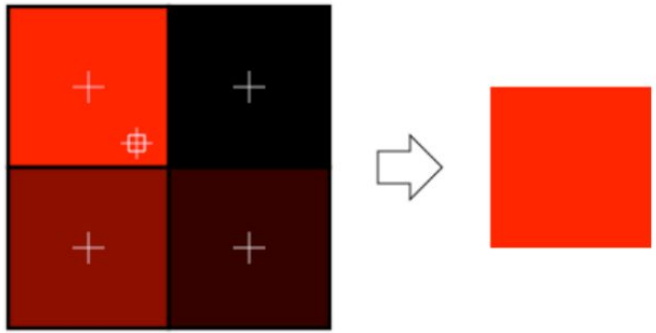


Image courtesy of opengl-tutorial.org

Filtering

- Nearest
- **Bilinear**
- Trilinear
- Anisotropic

Blend surrounding texels with weights proportional to sample's distance from center.

- GL_LINEAR

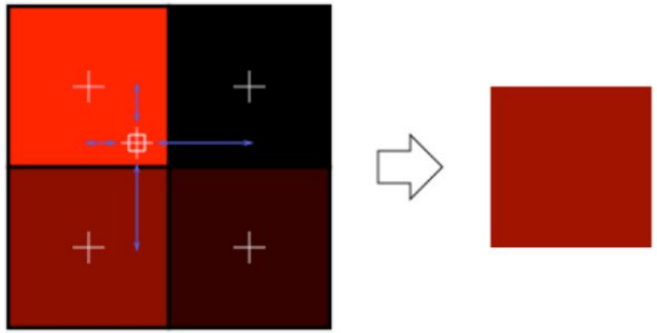


Image courtesy of opengl-tutorial.org

Filtering

- Nearest
- Bilinear
- **Trilinear**
- Anisotropic

The texture sample likely falls between two mipmap levels. Trilinear filtering does the following:

- Perform a bilinear sample on each mipmap level.
- Linearly blend the results.
- GL_LINEAR_MIPMAP_LINEAR

Filtering

- Nearest
- Bilinear
- Trilinear
- **Anisotropic**

Image courtesy of Project CARS

Anisotropy is distortion in texels on an object surface viewed at an oblique angle relative to the screen plane.

- Sample a trapezoidal region of texels.
- Trapezoid determined by viewing angle.

OpenGL Textures

- OpenGL works with texture units.
  - glActiveTexture sets the texture unit subsequent state calls will affect.
  - Accepts GL_TEXTURE*i*, where *i* is between 0 and GL_MAX_COMBINED_TEXTURE_IMAGE_UNITS.
- The unit index is bound to the shader program uniform.
  - glUniform1i
- Texture data is loaded by calling glTexImage2D.
- Mipmaps can be generated with glGenerateMipmaps.

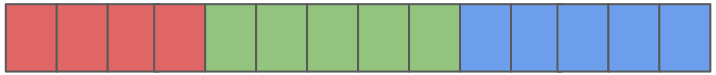
Worth Noting

- To save space, texture data is often compressed.
  - There are many compression algorithms: BC1-3, BC4, BC5, BC6H, BC7, ASTC, PVRTC
  - Hardware unpacks it for you!
  - This is one of those things that could be an entire lecture.
- Compression may leave artifacts. It’s a tradeoff.

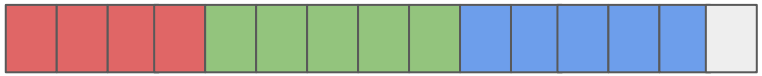
Texture Formats

GL_RGBA8

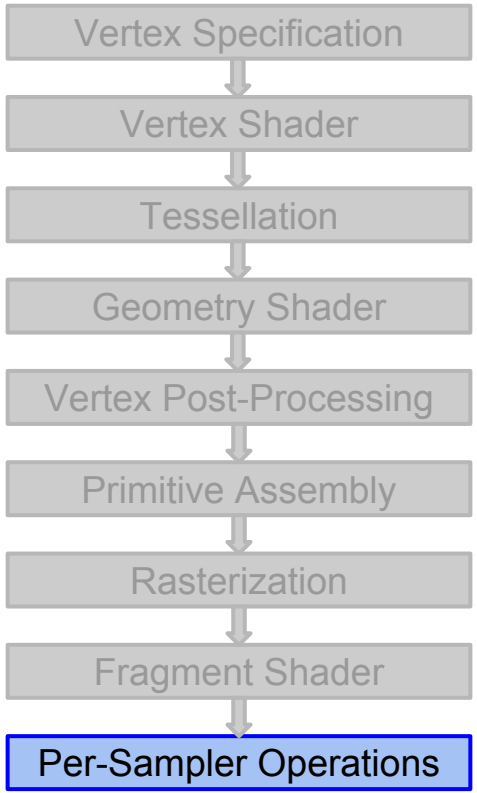
GL_RGB5



GL_RGB5_A1



Programmable Pipeline



Per-Sample Operations

- Blending
- sRGB Conversion
- Write Mask

We’re almost at the end of the graphics pipeline!

There are a few operations left that happen after the fragment shader and before the result is placed in the output framebuffer.

Per-Sample Operations

- **Blending**
- sRGB Conversion
- Write Mask

Image courtesy of MSDN

Blending is hardware functionality for blending the output fragment color with the existing color in the target framebuffer.

Blending is controlled by blend state, which consists of:

- Blending equation
- Parameters for the equation

Alpha is often treated independently.

Per-Sample Operations

- Blending
- **sRGB Conversion**
- Write Mask

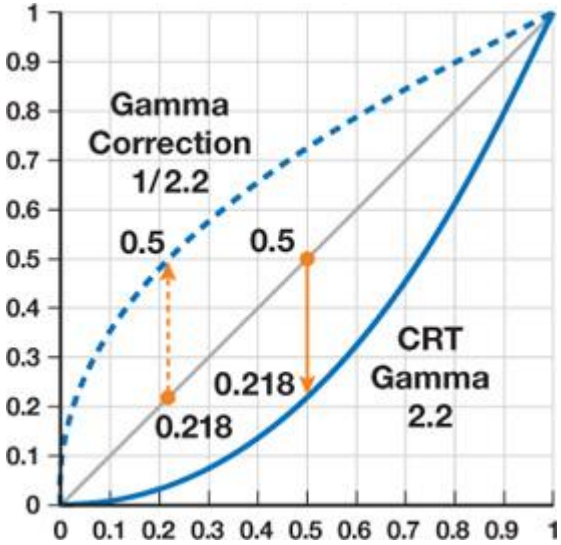


Image courtesy of Nvidia

Displays are nonlinear with respect to conversion of voltages to light intensity.

- Originated from CRTs. LCD displays simply mimic this behavior.

Conversion from linear RGB value to sRGB color space will ensure the displayed color closely matches the original value.

Per-Sample Operations

- Blending
- sRGB Conversion
- **Write Mask**

The write mask prevents the writing of specific color channels to the target framebuffer.

- In OpenGL, glColorMask.

For example…

- glColorMask(false, true, true, true) means that the red channel will not be written.

Swap Buffers

- We’re almost there!
- The last thing that needs to happen is the **buffer swap**.
  - We’ve been drawing our scene to the **back buffer**.
  - The **front buffer** is the image currently being displayed.
  - A swap command switches the front buffer and the back buffer, so our new image is displayed.
- In OpenGL, it varies how this is done.
  - We use SDL, which requires a call to SDL_GL_SwapWindow.
  - If you’re using wgl, you’d use wglSwapBuffers.

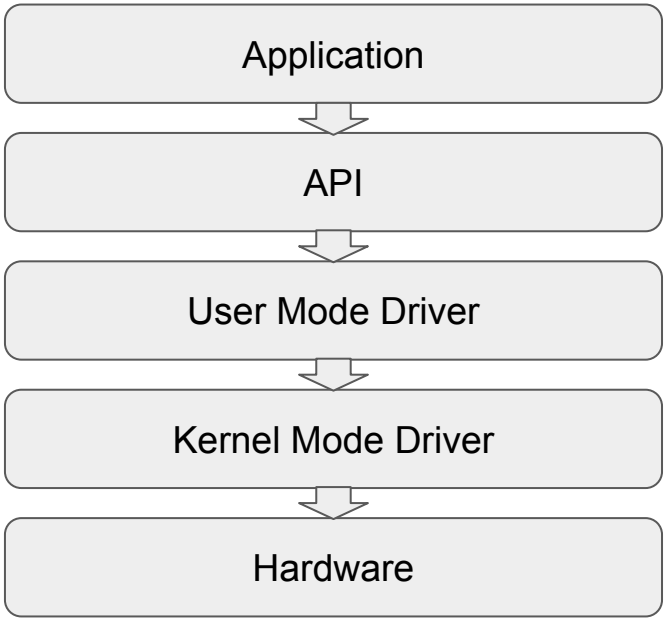
End of Pipeline (Lecture)

- Homework 3 is due 11:59pm eastern, Tuesday, 2/21.
- Want to learn more?
  - sRGB and the importance of linear-space lighting:
    - http://http.developer.nvidia.com/GPUGems3/gpugems3_ch24.html
  - Repeated from first lecture - a journey through the graphics pipeline:
    - https://fgiesen.wordpress.com/2011/07/09/a-trip-through-the-graphics-pipeline-2011-index/
  - Bresenham's line raster algorithm:
    - https://en.wikipedia.org/wiki/Bresenham's_line_algorithm

Bonus Material!

Software Layers

- There are multiple layers of software between your application and the hardware.



Vsync and Vblank

- Vsync is short for **vertical synchronization**.
  - It is the process of synchronizing the swapping of the front and back buffers with the displays vblank.
- Vblank is short for **vertical blanking interval**.
  - It’s the time during which the raster scan line is traced from the end of the final line of a frame back to the beginning of the next frame.
  - So, the raster scan line moves from bottom right, back to top left, assuming it’s scanning left-to-right, top-to-bottom.
- Vsync is intended to prevent tearing...

Tearing

Triple Buffering

- Instead of using just back and front buffers, add another buffer.
- Pros
  - Reduces dependencies between the GPU and CPU.
  - Further combats tearing.
  - Increased performance.
- Cons
  - Increased GPU memory usage.
  - Increased latency.