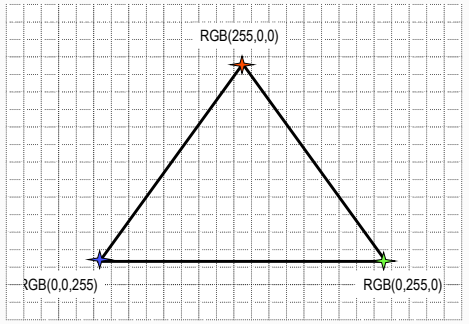
ECE4893A/CS4803MPG: Multicore and GPU Programming for Video Games

3D Rendering Pipeline (III)

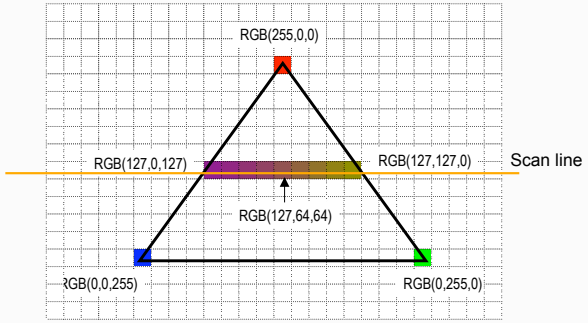

Prof. Hsien-Hsin Sean Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology

Rasterization: Shading a Triangle



- Converting geometry to a raster image (i.e., pixels)
- Paint each pixel's color (by calculating light intensity) on your display (Typically object-based)
- Gouraud Shading: Intensity Interpolation of vertices

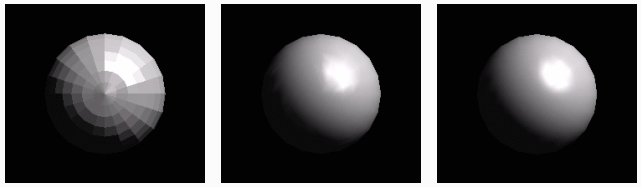
Gouraud Shading



- Scan conversion algorithm

Comparison of Shading Methods

Source: Michal Necasek

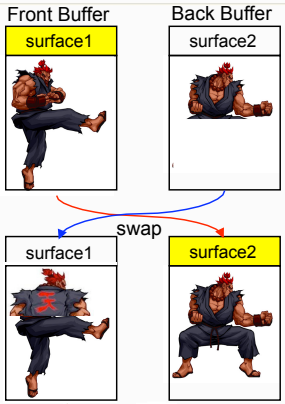


Flat shading | Gouraud shading | Phong shading

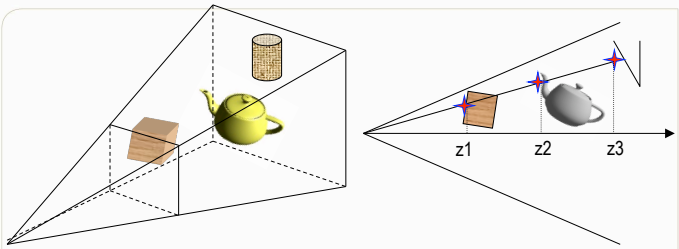
- Phong shading
  - requires generating per-pixel normals to compute light intensity for each pixel, not efficient for games
  - Can be done on GPGPU today using Cg or HLSL
- Gouraud shading is supported by Graphics hardware

Double Buffering

- Display refreshes at 60 ~ 75 Hz
- Rendering could be "faster" than the refresh period
- Too fast leads to
  - Frames not shown
- Too slow leads to
  - New and old frame mixed
  - Flickering
- Solution:
  - *Double or multiple buffering*



Z-Buffer

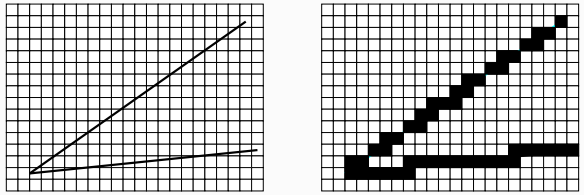


- Also called *depth buffer*
- Draw the pixel which is nearest to the viewer
- Number of the entries corresponding to the screen resolution (e.g. 1024x768 should have a 768k-entry Z-buffer)
- Granularity matters
  - 8-bit never used
  - 16-bit z value could generate artifacts

Z-Buffer Bandwidth Could Be an Issue

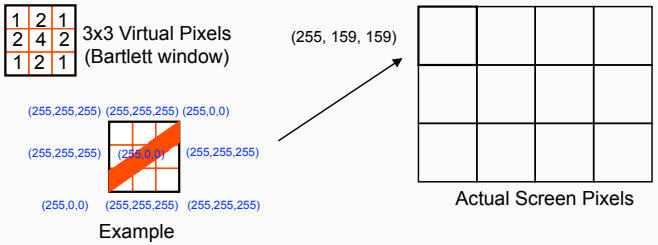
- Perform Z test before drawing a pixel
- How much bandwidth needed for RZ+WZ+RC+WC +TR per pixel?
- Overdrawn rate is about 3 out of 4
  - One every 4 pixels drawn is for clearing the Z-buffer
- Some cards perform Z-compression to alleviate bandwidth issues

Aliasing



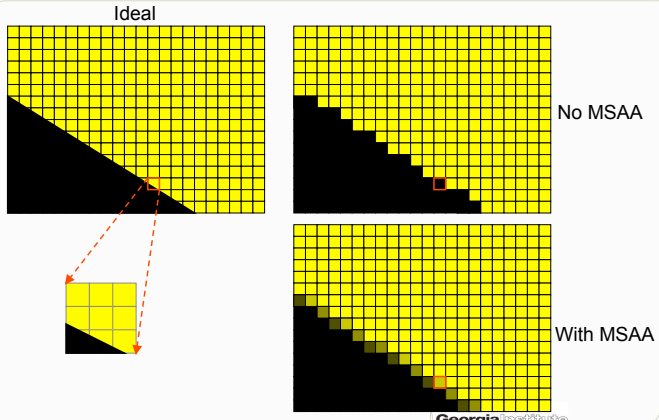
- Jagged line (or staircase)
- Can be improved by increasing resolution (i.e. more pixels)

Anti-Aliasing by Multisampling (Example: Supersampling)

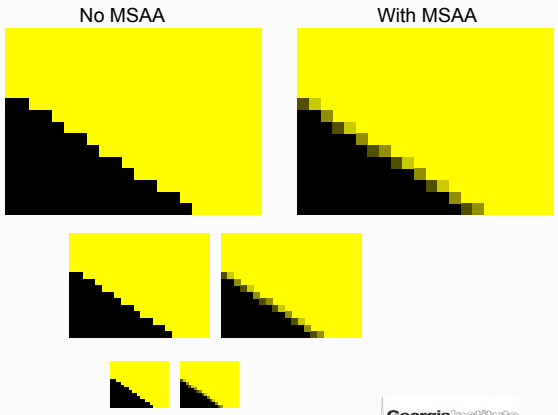


- GPU samples multiple locations for a pixel
- Several different methods
  - e.g., grid (as shown), random, GeForce's quincunx
- Downside
  - Blurry image
  - Increased memory (e.g., z-buffer) storage for subpixel information

Anti-Aliasing Example



Visualizing Anti-Aliasing Example



Texture Mapping

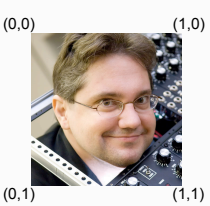
- Rendering tiny triangles is slow
- Players won't even look at some certain details
  - Sky, clouds, walls, terrain, wood patterns, etc.
- Simple way to add details and enhance realism
- Use 2D images to map polygons
- Images are composed of 2D "texels"
- Can be used to substitute or blend the lit color of a texture-mapped surface

Texture Mapping

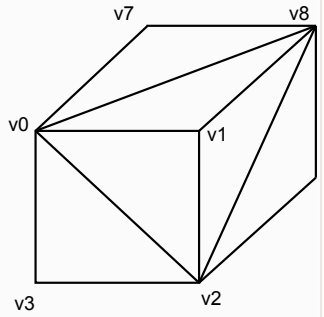
- Introduce one more component to geometry
  - Position coordinates
  - Normal vector
  - Color
  - *Texture coordinates*
- Texture info
  - (u, v) coordinates for each vertex
  - Typically range from 0 to 1
    - (0,0) from upper left corner

Georgia Institute of Technology

Texture Mapping



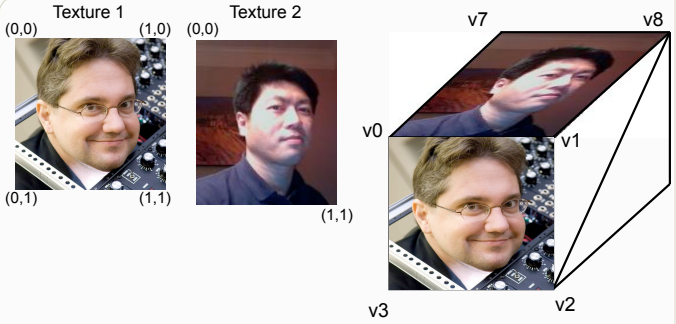
Texture: hunk.jpg




```
                    u  v
                    ↓  ↓
{v1.x, v1.y, v1.z, …, 1, 0},
{v2.x, v2.y, v2.z, …, 1, 1},
{v0.x, v0.y, v0.z, …, 0, 0},
{v3.x, v3.y, v3.z, …, 0, 1},
```

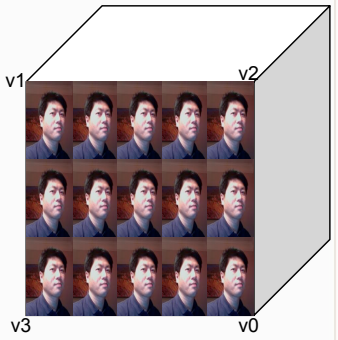
Georgia Institute of Technology

Texture Mapping



Georgia Institute of Technology

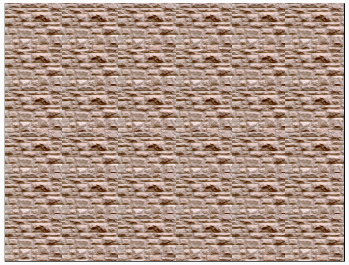
Texture Mapping




```
                    u  v
                    ↓  ↓
{v1.x, v1.y, v1.z, …, 0, 0},
{v2.x, v2.y, v2.z, …, 5, 0},
{v0.x, v0.y, v0.z, …, 5, 3},
{v3.x, v3.y, v3.z, …, 0, 3},
```

Georgia Institute of Technology

Texture Mapping

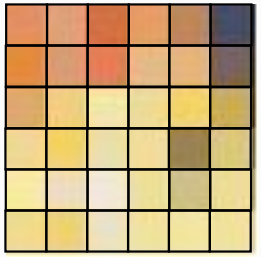


u v

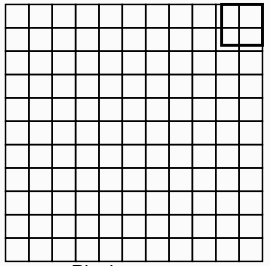
```
{v1.x, v1.y, v1.z, …, 0, 0},
{v2.x, v2.y, v2.z, …, 6, 0},
{v0.x, v0.y, v0.z, …, 6, 6},
{v3.x, v3.y, v3.z, …, 0, 6},
```

Georgia Institute of Technology

Magnification

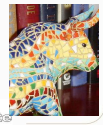


Texels



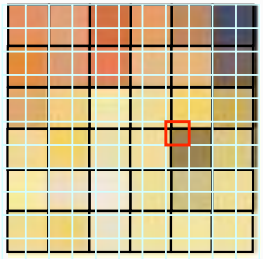
Pixels on screen

- Texel and pixel mapping is rarely 1-to-1
- Mapped triangle is very close to the camera
- One texel maps to multiple pixels

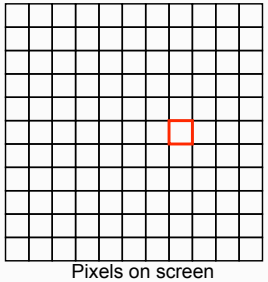


Georgia Institute of Technology

Nearest Point Sampling (for Magnification)



Texels

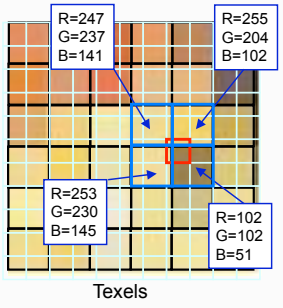


Pixels on screen

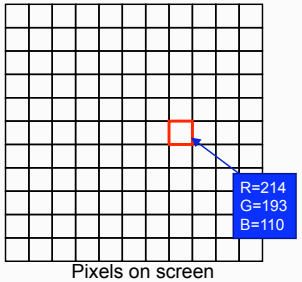
- Choose the texel nearest the pixel's center

Georgia Institute of Technology

Bi-linear Filtering (for Magnification)



Texels

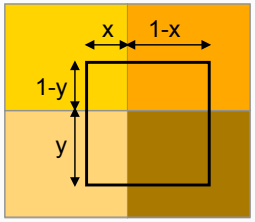


Pixels on screen

- Average for the 2x2 texels surrounding a given pixel

Georgia Institute of Technology

Bi-linear Filtering (for Magnification)



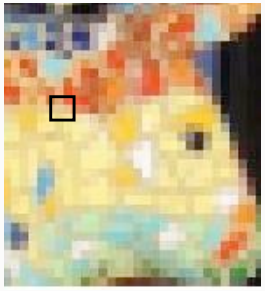
* (1-x) * (1-y)
* (1-x) * y
* x * (1-y)
+) * x * y

Final Color

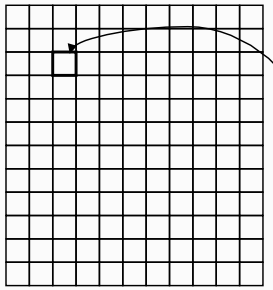
: pixel enclosed by 4 texels

- Or take the weighted color values for the 2x2 texels surrounding a given pixel

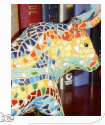
Minification



Texels

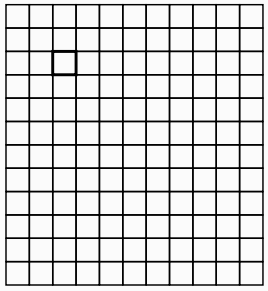


Pixels on screen



- Texel and pixel mapping is rarely 1-to-1
- Multiple texels map to one pixel

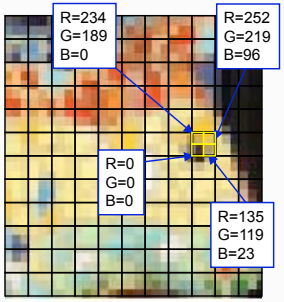
Nearest Point Sampling (for Minification)

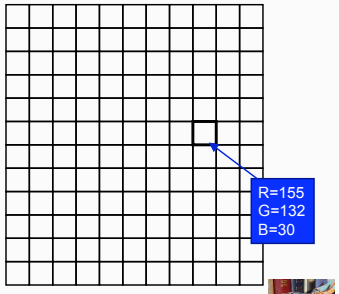


Pixels on screen

- Choose the texel nearest the pixel's center

Bi-linear Filtering (for Minification)





Pixels on screen

- Average for the 2x2 texels surrounding a given pixel

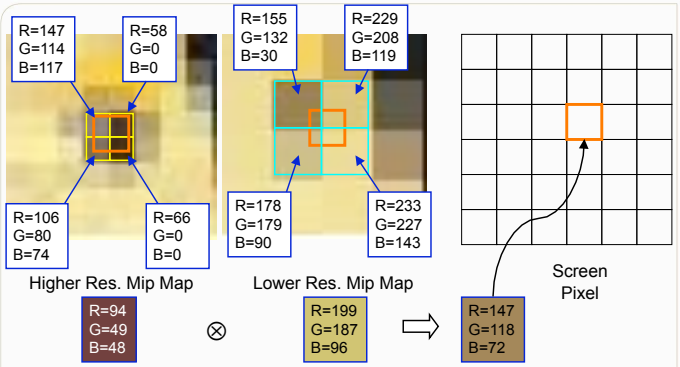
Mip-mapping

- Multiple versions are provided for the same texture
- Different versions have different levels of details
  - E.g., 7 LOD maps: 256x256, 128x128, 64x64, 32x32, 16x16, 8x8, 4x4
  - Choose the closet maps to render a surface
- Maps can be automatically generated by 3D API
- Accelerate texture mapping for far-away polygons
- More space to store texture maps

Mip-mapping

- API or Hardware can
  - Generate (lower resolution) mip maps automatically
  - Choose the right one for the viewer
    - Good performance for far triangles
    - Good LOD for close-by objects
  - Tri-linearly interpolate

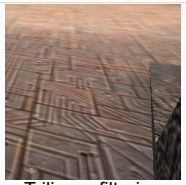
Tri-linear Filtering using Mip maps



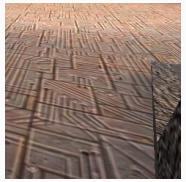
- Interpolate between mipmaps

Anisotropic Filtering


Bilinear filtering


Trilinear filtering


16x Anisotropic filtering


64x Anisotropic filtering

- Not isotropic
- Preserving details for oblique viewing angles (non-uniform surface)
- AF calculates the "*shape*" of the surface before mapping
- The number of pixels sampled depends on the distance and view angles relative to the screen
- Very expensive

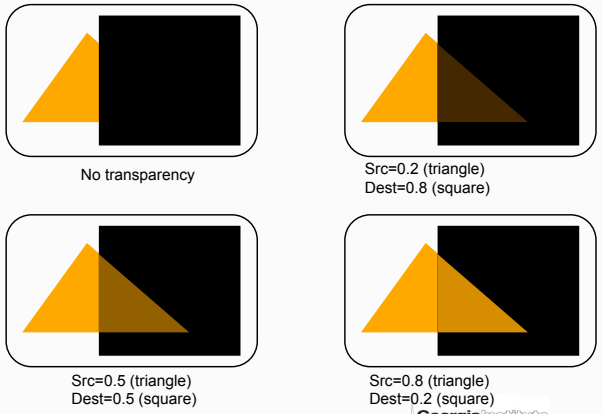
Source: nvidia

Color Blending and Alpha Blending

- Transparency effect (e.g. Water, glasses, etc.)
- Source color blended with destination color
- Several blending methods
  - Additive
    C = SrcPixel ⊗ (1,1,1,1) + DstPixel ⊗ (1,1,1,1) = SrcPixel + DstPixel
  - Subtractive
    C = SrcPixel ⊗ (1,1,1,1) — DstPixel ⊗ (1,1,1,1) = SrcPixel — DstPixel
  - Multiplicative
    C = DstPixel ⊗ SrcPixel
  - Using Alpha value in the color (Alpha Blending)
    C = SrcPixel ⊗ (α,α,α,α) + DstPixel ⊗ (1-α,1-α,1-α,1-α)
  - And many more in the API …

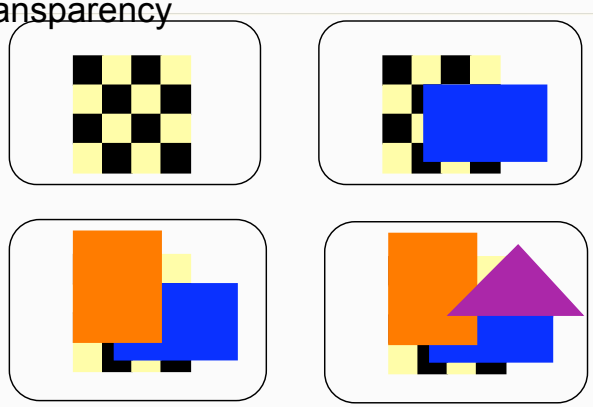
Georgia Institute of Technology

Alpha Blending (Inverse Source form)



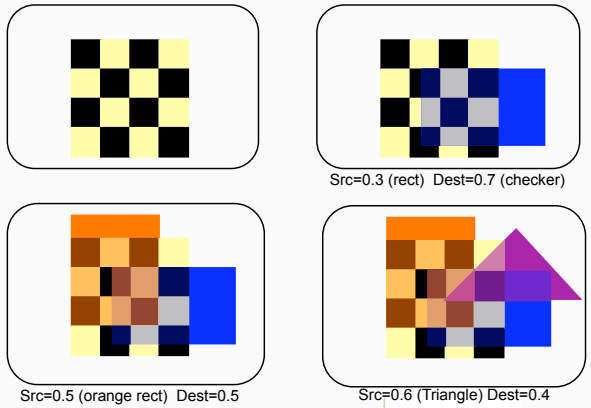
Georgia Institute of Technology

Another Example Without Transparency



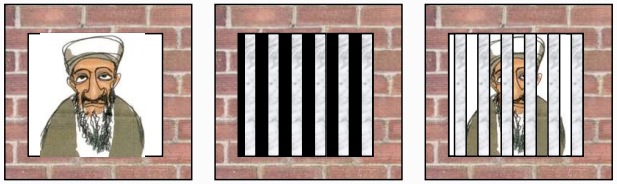
Georgia Institute of Technology

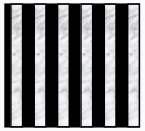
Another Alpha Blending Example



Georgia Institute of Technology

Alpha Test





Texture: bar.jpg

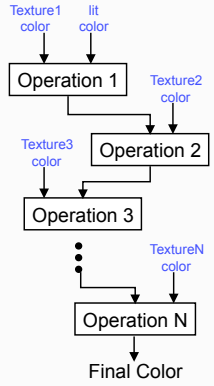
Straightforward texture mapping

```
if (α op val)
  reject pixel
else
  accept pixel
```

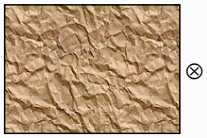
- Reject pixels by checking their alpha values
- Model fences, chicken wires, etc.

Multi-Texturing

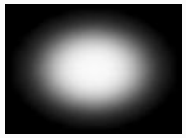
- Map multiple texture to a polygon
  - Common APIs support 8 textures
- Performance will be reduced
- Multiple texturing stages in the pipeline
- Texture color will be calculated by
  - Multiplication
  - Addition
  - Subtraction



Multi-Texturing Example: Light Mapping



Some crumpled paper texture



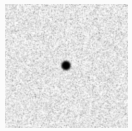
A spotlight map

Different alpha blending

Bump Mapping

- *Per-fragment lighting* using bump map (normal map) to perturb surface normal
- No geometry tessellation, avoid geometry complexity
- Store normal vectors rather than RGB colors for bump map
- Apply per-pixel shading (w/ light vector, e.g., Phong Shading)

Shaded sphere



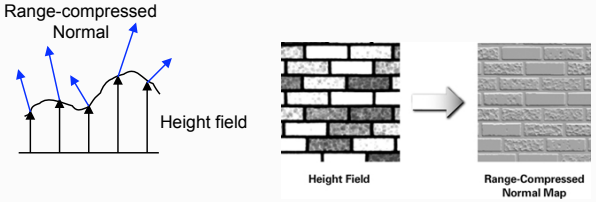
Bump map

Bump mapped sphere

Source: wikipedia

Bump Mapping

- Normal map was derived from a Height field map
  - Height field stores the "elevation" for each texel
  - Sample texel's height as well as texels to the right and above



(Cube) Environment Mapping

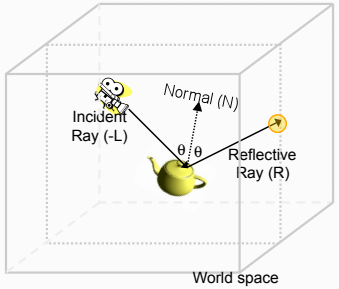
- Cube Map Textures (in *World coordinate*)
- Each face encodes 1/6 of the panoramic environment

Source: Game Creators, Ltd.

Reflective (or Environment) Mapping

Source: zabur.smz.sk

- Look up the environment map
- Add reflection to a fragment's final color



Function texCUBE() is provided for sampling the texel in cube map

R = 2*(dotp(N, L))*N - L

Reflective Mapping

Cube texture map

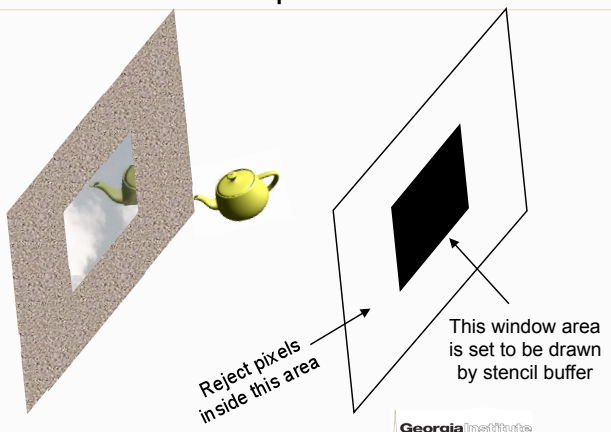
Rendered image

Stenciling

- Stencil buffer
  - To reject certain pixels to be displayed
  - To create special effect similar to alpha test
    - Mask out part of the screen
  - Set together with Z-buffer in 3D API
  - Perform prior to Z-buffer test

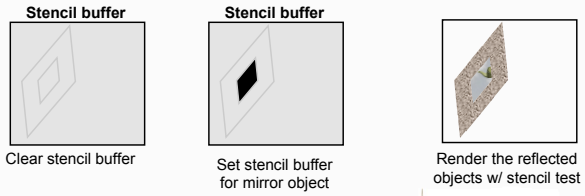
```
if ((stencil ref & mask)
      op (pixel val & mask))
  accept pixel
else
  reject pixel
```


Stencil Buffer Example



Mirror Effect

1. Render the entire scene as normal (no reflection yet)
2. Clear the entire stencil buffer to '0' (i.e., mirror's fragments)
3. Render the mirror primitives and set the corresponding stencil buffer fragment to '1'
4. Render the reflected objects only if stencil test passes (i.e., value==1)
   - Using a "reflection matrix" for world transformation (Draw the scene as if they are seen in the mirror)



Mirror Effect (Cont.)

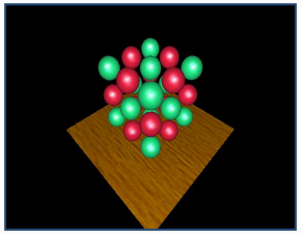
Can be done in a reverse order

1. Render the reflected image of the scene using a "reflection matrix" for world transformation (Draw the scene as if they are seen in the mirror)
2. Render non-reflected with stencil buffer accept/reject test to prevent the reflected image being drawn over

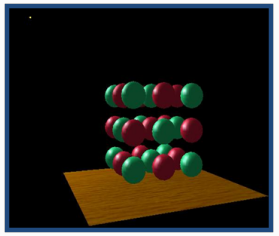
Cast Shadow

- Light source is the eye
- Objects are not visible from the light are in shadow (but illuminated by ambient or emissive lights)
- Two-pass process proposed by Lance Williams in SIGGRAPH'98

Shadow Map



Scene from light's Point-of-view

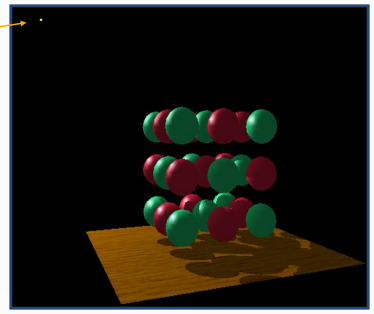


Scene from eye's Point-of-view

Figure Source: Justin Reynolds

Shadow Map

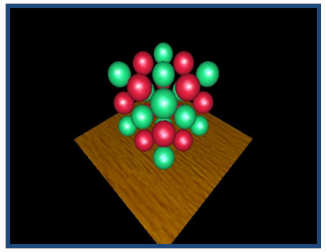
Light source

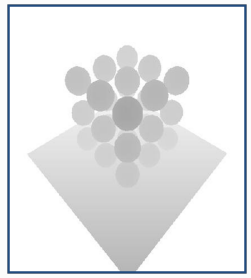


Desired rendered scene with shadow

Figure Source: Justin Reynolds

Shadow Map





Depth buffer (or shadow map) from the **light's** point-of-view

Figure Source: Justin Reynolds

Shadow Map Concept

Compare z' with z_M = shadow_map(x', y')
(the z' is deeper than z_M in this example)

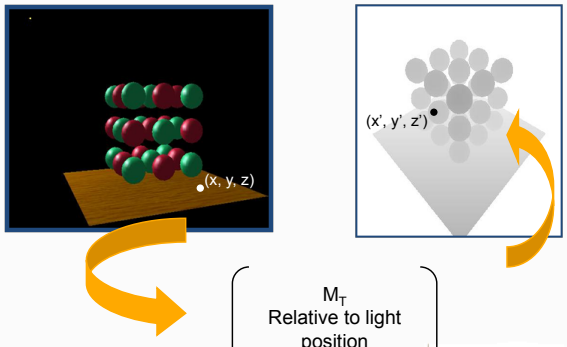


Figure Source: Justin Reynolds

Shadow Map

Depth testing from the light's position

Two pass algorithms

1. Render shadow map (depth buffer) from the light's point of view to Indicate the depth of the closest pixels to the light
   - Using specialized vertex and pixel shaders
   - Pixel shader writes pixel depth (not color)

2. Render scene from viewer's position, for each rasterized fragment
   - Determine fragment's XYZ position relative to the light
   - Compare the fragment's depth and the depth stored in the shadow map
   - The pixel is in shadow if P_z > shadow_map(p_x, p_y)
   - Can only show ambient+emissive light color for shadowed pixel

Shadow Map Algorithm

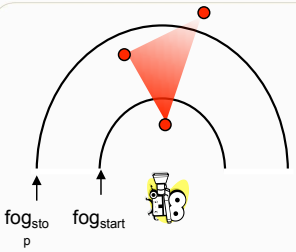
```
procedure SHADOWMAPPING

Render depth buffer (Z-buffer) from lights point of view,
  resulting in a shadow map or depth map
Now, render scene from the eye's point of view

for all rasterized fragments do
    Determine fragment's xyz position relative to the light
     That is transform each fragment's xyz into the light's coordinate
system
    A = depth map(x,y)
    B = z-value of fragment's xyz light position
    if A < B then
        fragment is shadowed
    else
        fragment is lit
    end if
end for
```

Algorithm Source: Justin Reynolds

Apply Fog Effect



$$color = (1 - f) \cdot Color_{vertex} + f \cdot Color_{fog}$$

$$f = MAX\left(\frac{dist(eye, vertex) - fog_{start}}{fog_{stop} - fog_{start}}, 0\right)$$

- Provide depth cue
  - Simulate weather condition
  - Avoid *popping* effect
- Color blending

- Calculate distance
- Calculate intensity of vertex color based on distance
  - Color blending
  - Linear density, exponential density
- Blending color schemes
  - Per-vertex (then Interpolate pixels), less expensive
  - Per-fragment basis (Nvidia hardware), better quality