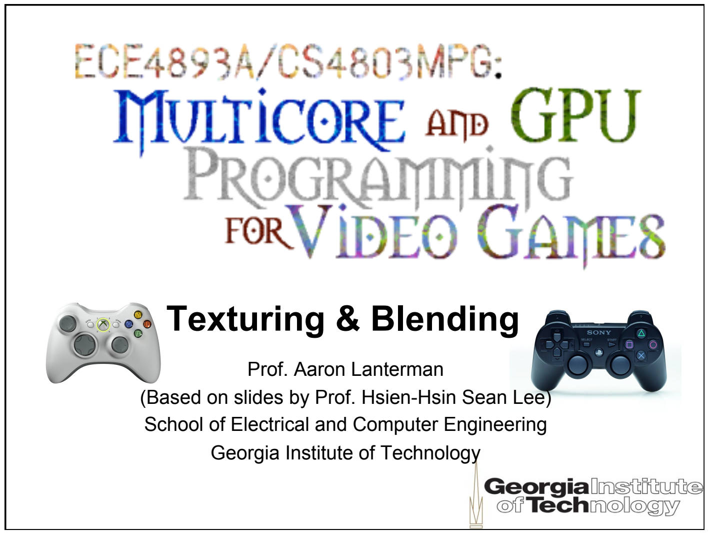
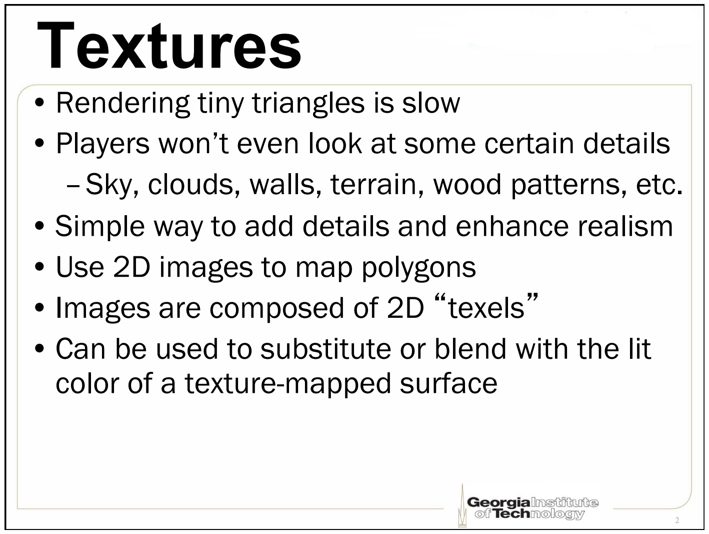


ECE4893A/CS4803MPG: Multicore and GPU Programming for Video Games

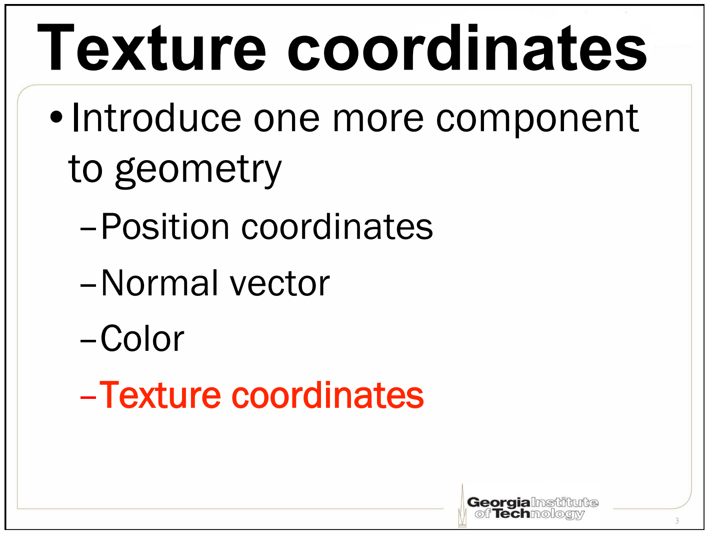
Texturing & Blending

Prof. Aaron Lanterman
(Based on slides by Prof. Hsien-Hsin Sean Lee)
School of Electrical and Computer Engineering
Georgia Institute of Technology



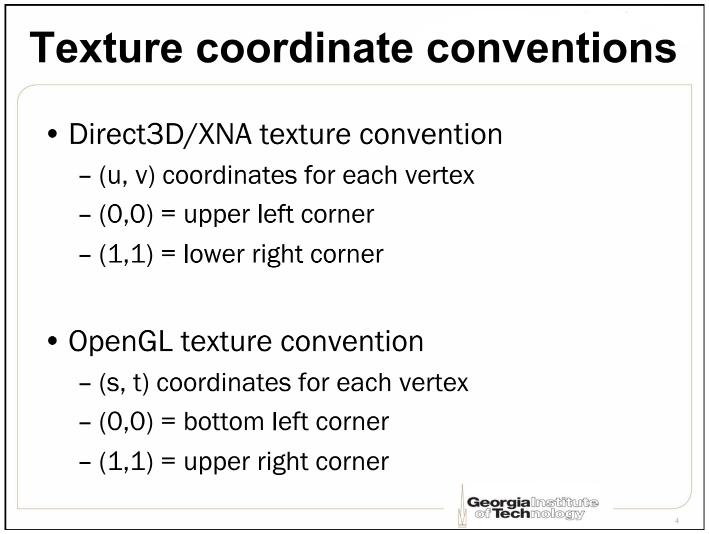
Textures

- 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 with the lit color of a texture-mapped surface



Texture coordinates

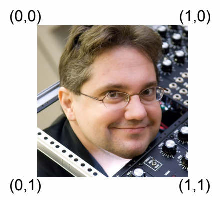
- Introduce one more component to geometry
  - Position coordinates
  - Normal vector
  - Color
  - Texture coordinates

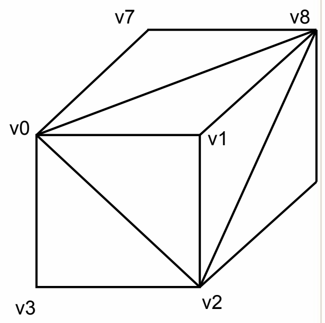


Texture coordinate conventions

- Direct3D/XNA texture convention
  - (u, v) coordinates for each vertex
  - (0,0) = upper left corner
  - (1,1) = lower right corner
- OpenGL texture convention
  - (s, t) coordinates for each vertex
  - (0,0) = bottom left corner
  - (1,1) = upper right corner

Texture mapping example (1)



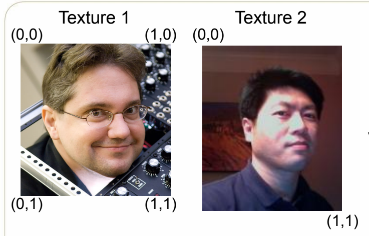


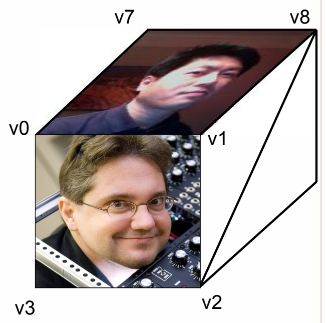
Direct3D/XNA convention

u v

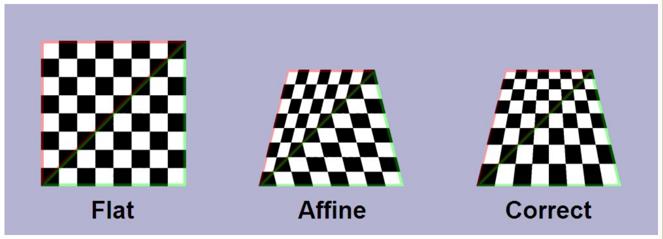
```
{v1.x, v1.y, v1.z, …, 1, 0},
{v2.x, v2.y, v2.z, …, 1, 1},
{v0.x, v0.y, v0.z, …, 0, 0},
{v3.x, v3.y, v3.z, …, 0, 1},
```

Texture mapping example (2)



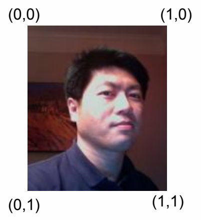


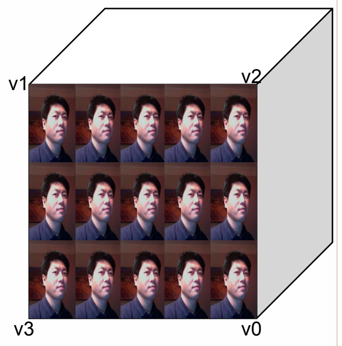
“Perspective correct” texture mapping



From http://en.wikipedia.org/wiki/Texture_mapping

Repeated textures

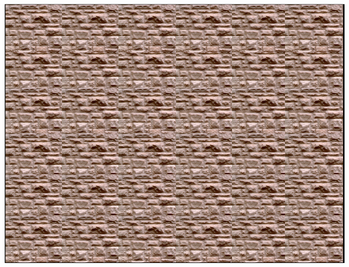




u v

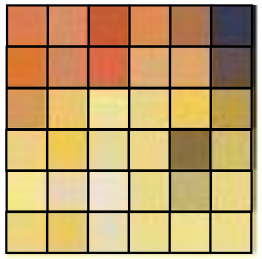
```
{v1.x, v1.y, v1.z, …, 0, 0},
{v2.x, v2.y, v2.z, …, 5, 0},
{v0.x, v0.y, v0.z, …, 5, 3},
{v3.x, v3.y, v3.z, …, 0, 3},
```

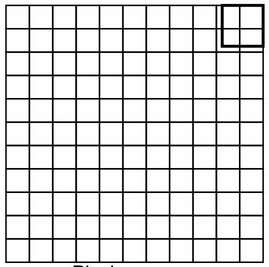
Repeated brick texture



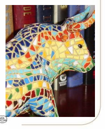
```
                              u  v
                              ↓  ↓
{v1.x, v1.y, v1.z, …, 0, 0},
{v2.x, v2.y, v2.z, …, 6, 0},
{v0.x, v0.y, v0.z, …, 6, 6},
{v3.x, v3.y, v3.z, …, 0, 6},
```

Magnification

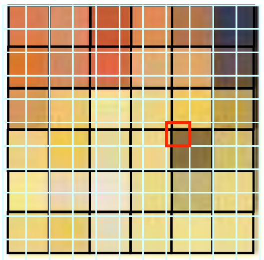

Texels

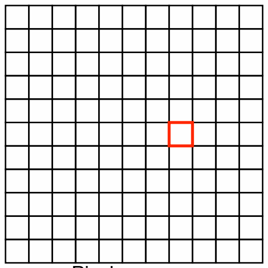

Pixels on screen

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



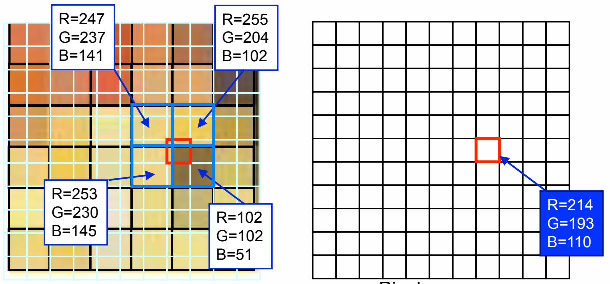
Nearest point sampling (for magnification)


Texels


Pixels on screen

- Choose the texel nearest the pixel's center

Averaging (for magnification)

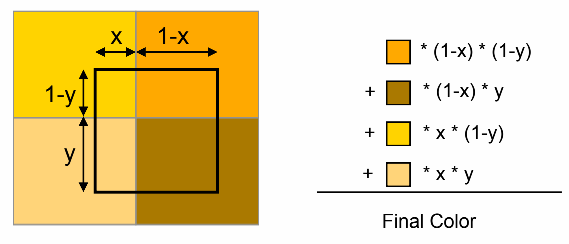


Texels

Pixels on screen

- Average the 2x2 texels surrounding a given pixel

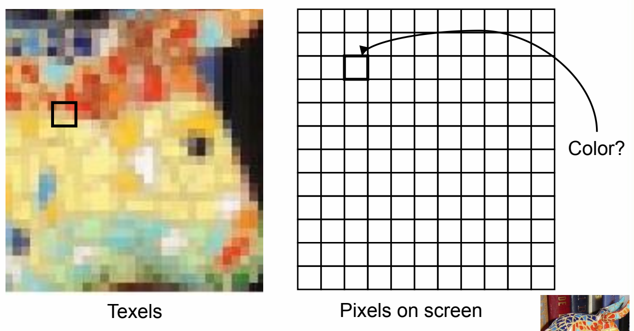
Bilinear filtering (for magnification)

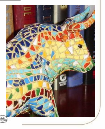


□ : pixel enclosed by 4 texels

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

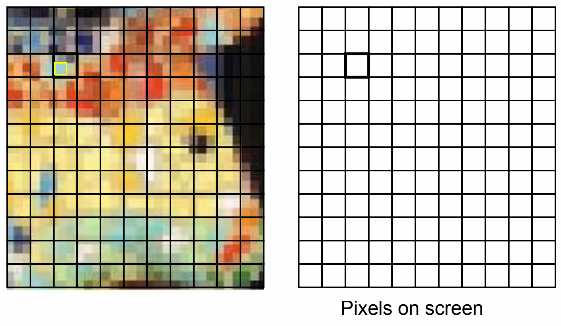
Minification

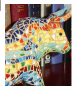




- Texel and pixel mapping is rarely 1-to-1
- Multiple texels map to one pixel

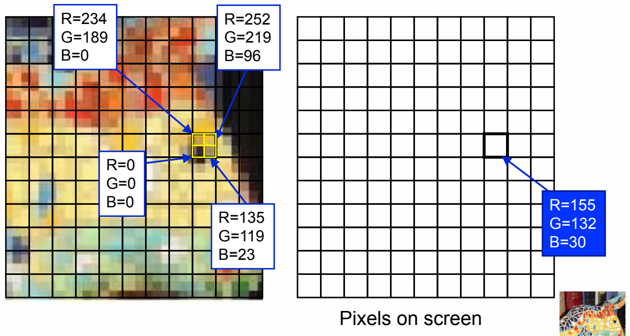
Nearest point sampling (for minification)

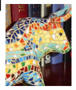




- Choose the texel nearest the pixel's center

Averaging (for minification)



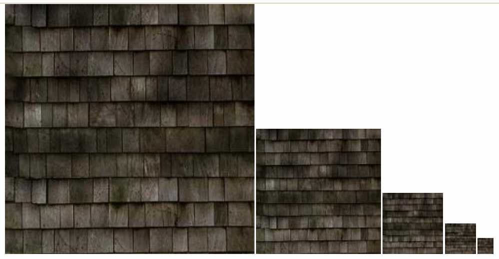


- Average for the 2x2 texels corresponding to a given pixel

Mip-mapping (1)

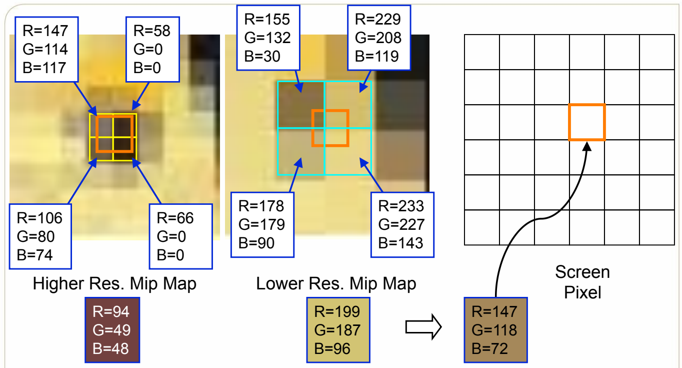
- Multiple versions are provided for the same texture
- Different versions have different levels of details
  - E.g., 7 LOD maps: 256x256, 128x128, 64x64, 32x32, 16x16, 8x8, 4x4
  - Choose the closest maps to render a surface
- Maps can be automatically generated by 3D API

Mip-mapping (2)



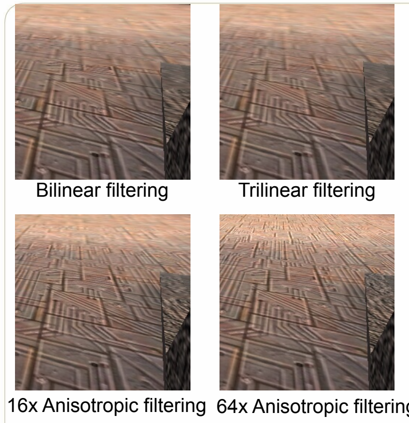
- API or hardware can
  - Choose the right one for the viewer
    - Good performance for far triangles
    - Good LOD for close-by objects
  - Trilinearly interpolate

Tri-linear filtering using mipmaps



- Interpolate between mipmaps

Anisotropic filtering



- Not isotropic
- Preserves 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

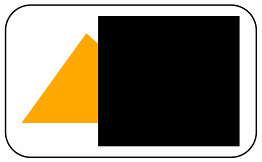
Source: nvidia

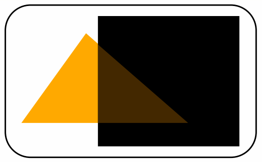
Color blending and alpha blending

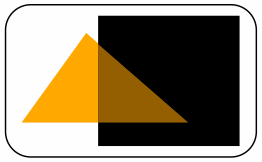
- Transparency effect (e.g. water, glasses, etc.)
- Source color blended with destination color
- Several blending methods
  - Additive
    C = SrcPixel ⊗ (1,1,1,1) + DstPixel ⊗ (1,1,1,1) = SrcPixel + DstPixel
  - Subtractive
    C = SrcPixel ⊗ (1,1,1,1) — DstPixel ⊗ (1,1,1,1) = SrcPixel — DstPixel
  - 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 …

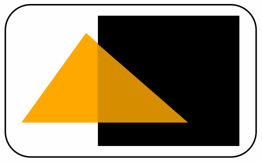
Georgia Institute of Technology 21

Alpha blending (inverse source form)


No transparency

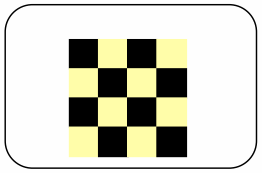
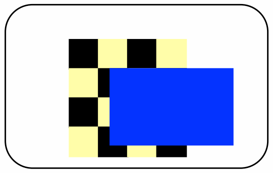
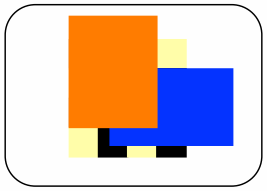
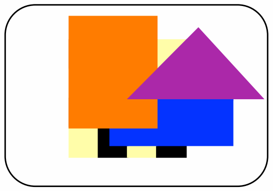

Src=0.2 (triangle)
Dest=0.8 (square)


Src=0.5 (triangle)
Dest=0.5 (square)


Src=0.8 (triangle)
Dest=0.2 (square)

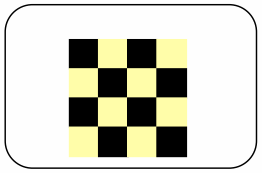
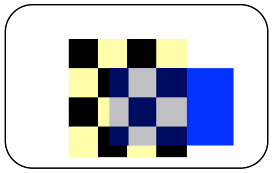
Georgia Institute of Technology 22

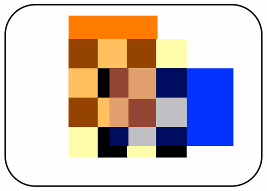
Another example w/out transparency

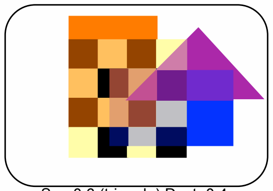





Georgia Institute of Technology 23

Another alpha blending example

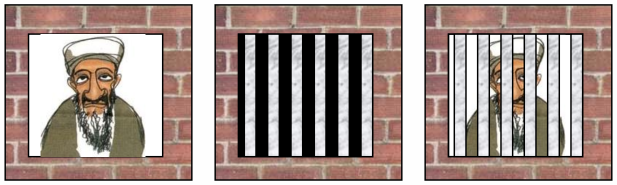


Src=0.3 (rect) Dest=0.7 (checker)

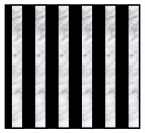

Src=0.5 (orange rect) Dest=0.5


Src=0.6 (triangle) Dest=0.4

Georgia Institute of Technology 24

Alpha test





Texture: bar.jpg

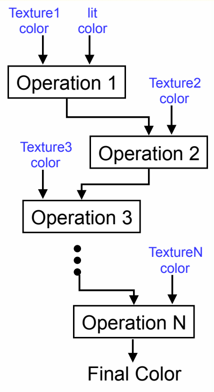
Straightforward texture mapping

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

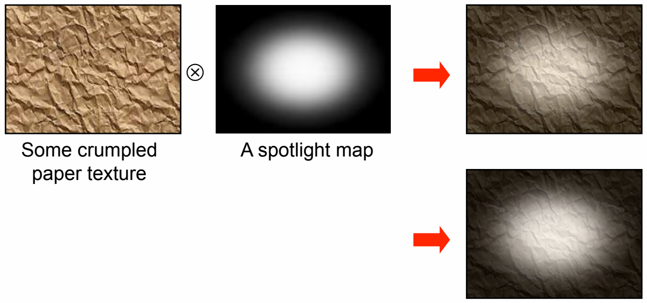
- Reject pixels by checking their alpha values
- Model fences, chicken wires, etc.

Multitexturing

- Map multiple textures 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

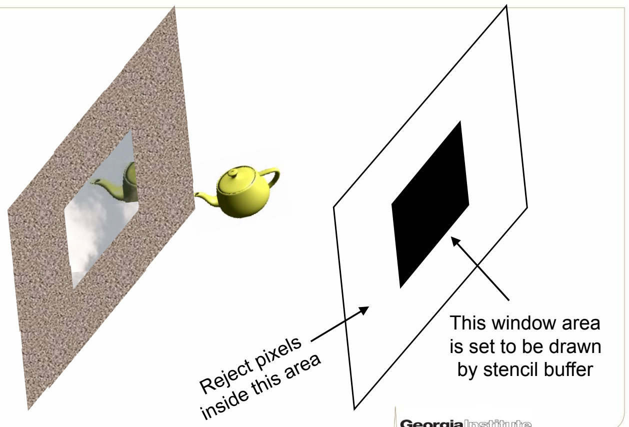
Different alpha blending

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
```

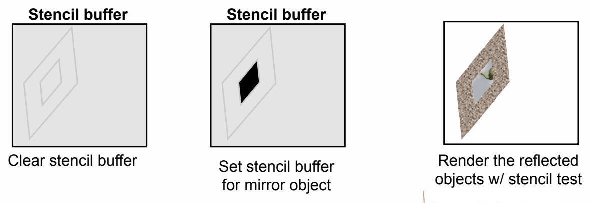
Stencil buffer example


Reject pixels inside this area
This window area is set to be drawn by stencil buffer

From http://www.ziggyware.com/readarticle.php?article_id=116

Mirror effect (1)

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)


Stencil buffer
Clear stencil buffer
Stencil buffer
Set stencil buffer for mirror object
Render the reflected objects w/ stencil test

From http://www.ziggyware.com/readarticle.php?article_id=116

Mirror effect (2)

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

From http://www.ziggyware.com/readarticle.php?article_id=116