Multi-Pass Rendering

- The pipeline takes one triangle at a time, so only local information, and pre-computed maps, are available
- Multi-Pass techniques render the scene, or parts of the scene, multiple times
  - Makes use of auxiliary buffers to hold information
  - Make use of tests and logical operations on values in the buffers
  - Really, a set of functionality that can be used to achieve a wide range of effects
    - Mirrors, shadows, bump-maps, anti-aliasing, compositing, …

Buffers

- Color buffers: Store RGBA color information for each pixel
  - OpenGL actually defines four or more color buffers: front/back, left/right and auxiliary color buffers
- Depth buffer: Stores depth information for each pixel
- Stencil buffer: Stores some number of bits for each pixel
- Accumulation buffer: Like a color buffer, but with higher resolution and different operations
- Buffers are defined by:
  - The type of values they store
  - The logical operations that they influence
  - The way they are written and read

Fragment Tests

- A fragment is a pixel-sized piece of shaded polygon, with color and depth information
- The tests and operations performed with the fragment on its way to the color buffer are *essential to understanding multi-pass techniques*
- Most important are, in order:
  - Alpha test
  - Stencil test
  - Depth test
  - Blending
- As the fragment passes through, some of the buffers may also have values stored into them

Alpha Test

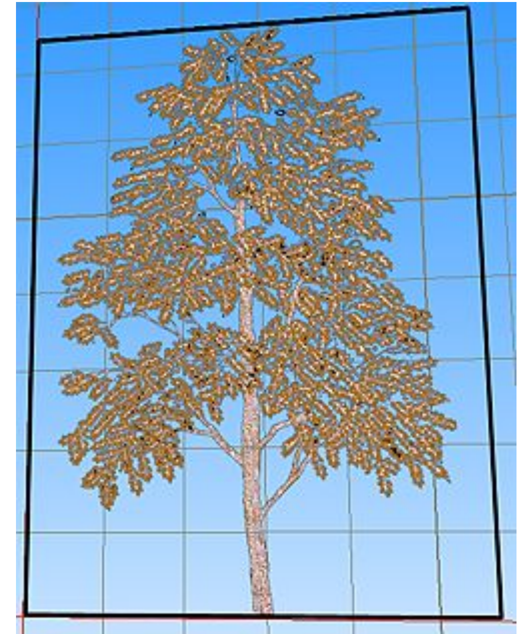
- The alpha test either allows a fragment to pass, or stops it, depending on the outcome of a test:

$$\texttt{if (} \alpha_{\texttt{fragment}} \texttt{ op } \alpha_{\texttt{reference}} \texttt{)}$$
$$\texttt{pass fragment on}$$

- Here, $\alpha_{\texttt{fragment}}$ is the fragment's alpha value, and $\alpha_{\texttt{reference}}$ is a reference alpha value that you specify
- `op` is one of:
  - <, <=, =, !=, >, >=
- There are also the special tests: Always and Never
  - Always let the fragment through or never let it through
- What is a sensible default?

Billboards

- Billboards are polygons with an image textured onto them, typically used for things like trees
  - More precisely, and image-based rendering method where complex geometry (the tree) is replaced with an image placed in the scene (the textured polygon)
- The texture normally has alpha values associated with it: 1 where the tree is, and 0 where it isn't
  - So you can see through the polygon in places where the tree isn't



Alpha Test and Billboards

- You can use texture blending to make the polygon see through, but there is a big problem
  - What happens if you draw the billboard and then draw something behind it?
  - Hint: Think about the depth buffer values
  - This is one reason why transparent objects must be rendered back to front
- The best way to draw billboards is with an alpha test: Do not let alpha < 0.5 pass through
  - Depth buffer is never set for fragments that are see through
  - Doesn't work for transparent polygons - more later

Stencil Buffer

- The stencil buffer acts like a paint stencil - it lets some fragments through but not others
- It stores multi-bit values
- You specify two things:
  - The test that controls which fragments get through
  - The operations to perform on the buffer when the test passes or fails
  - **All tests/operation look at the value in the stencil that corresponds to the pixel location of the fragment**
- Typical usage: One rendering pass sets values in the stencil, which control how various parts of the screen are drawn in the second pass

Stencil Tests

- You give an operation, a reference value, and a mask
- Operations:
  - Always let the fragment through
  - Never let the fragment through
  - Logical operations between the reference value and the value in the buffer: <, <=, =, !=, >, >=
- The mask is used to select particular bit-planes for the operation
  - `(reference & mask ) op ( buffer & mask )`

Stencil Operations

- Specify three different operations
  - If the stencil test fails
  - If the stencil passes but the depth test fails
  - If the stencil passes and the depth test passes
- Operations are:
  - Keep the current stencil value
  - Zero the stencil
  - Replace the stencil with the reference value
  - Increment the stencil
  - Decrement the stencil
  - Invert the stencil (bitwise)

Depth Test and Operation

- Depth test compares the depth of the fragment and the depth in the buffer
  - Depth increase with greater distance from viewer
- Tests are: Always, Never, <, <=, =, !=, >, >=
- Depth operation is to write the fragments depth to the buffer, or to leave the buffer unchanged
  - Why do the test but leave the buffer unchanged?
- **Each buffer stores *different* information about the pixel, so a test on one buffer may be useful in managing another**

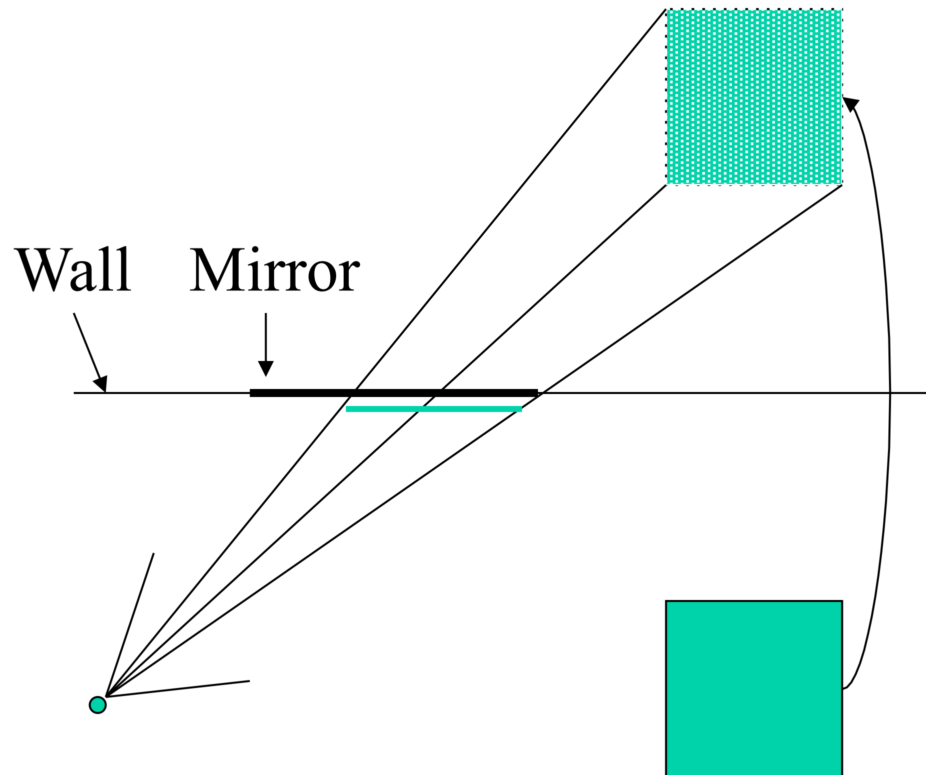
Multi-Pass Algorithms

- Designing a multi-pass algorithm is a non-trivial task
  - At least one person I know of has received a PhD for developing such algorithms
- References for multi-pass algorithms:
  - The OpenGL Programming guide discusses many multi-pass techniques in a reasonably understandable manner
  - Game Programming Gems has some
  - Watt and Policarpo has others
  - Several have been published as academic papers
  - As always, the web is your friend

Planar Reflections (Flat Mirrors)

- Use the stencil buffer, color buffer and depth buffer
- Basic idea:
  - We need to draw all the stuff around the mirror
  - We need to draw the stuff in the mirror, reflected, without drawing over the things around the mirror
- Key point: You can reflect the viewpoint about the mirror to see what is seen in the mirror, or you can reflect the world about the mirror

Reflecting Objects



- If the mirror passes through the origin, and is aligned with a coordinate axis, then just negate appropriate coordinate
- Otherwise, transform into mirror space, reflect, transform back

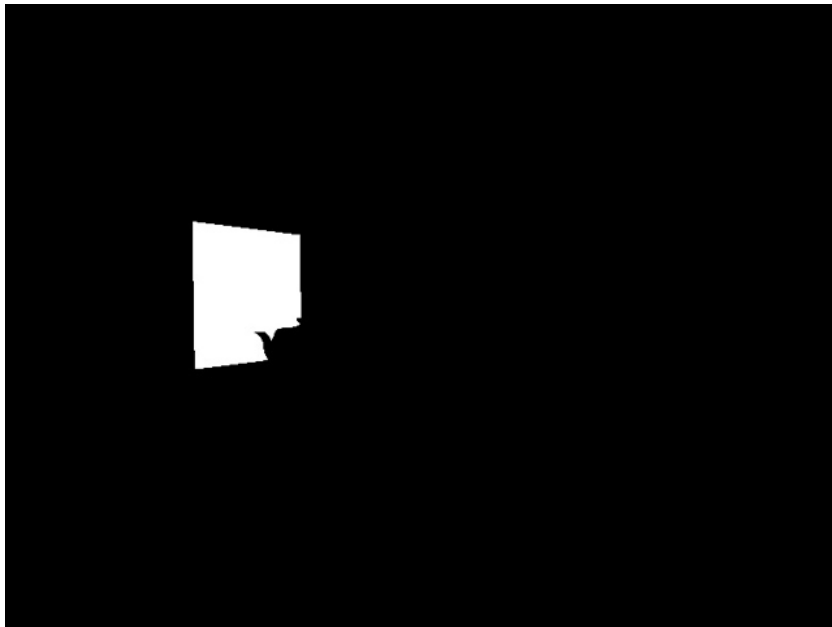
Small Problem

- Reflecting changes the apparent vertex order as seen by the viewer
  - Impacts back-face culling, so turn it off or change interpretation of vertex ordering
- Reflecting the view has the same effect, but this time it also shift the left-right sense in the frame buffer
  - Works, just harder to understand what's happening

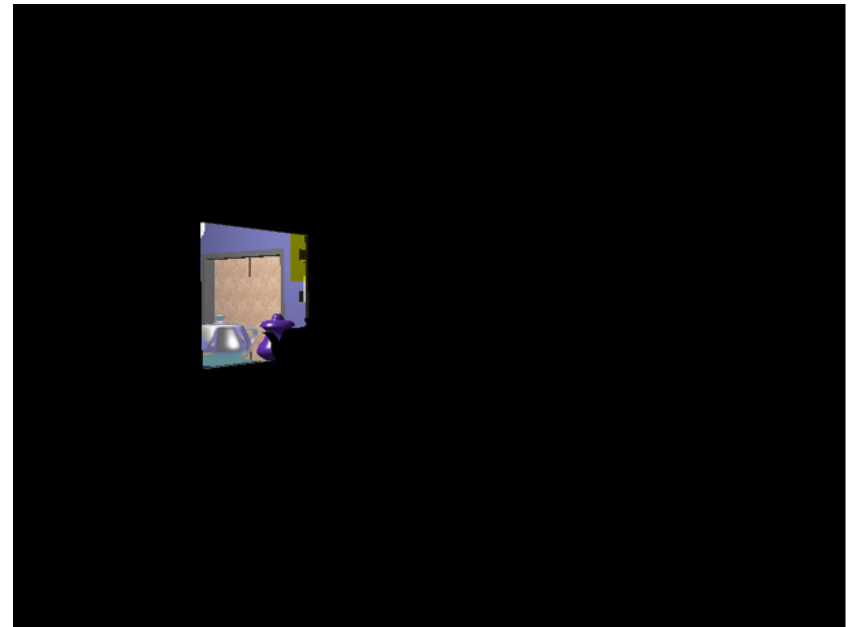
Rendering Reflected First

- First pass:
  - Render the reflected scene without mirror, depth test on
- Second pass:
  - Disable the color buffer, Enable the stencil buffer to always pass but set the buffer, Render the mirror polygon
  - Now, set the stencil test to only pass points outside the mirror
  - Clear the color buffer - does not clear points inside mirror area
- Third Pass:
  - Enable the color buffer again, Disable the stencil buffer
  - Render the original scene, without the mirror
  - Depth buffer stops from writing over things in mirror

Reflection Example



The stencil buffer after the second pass



The color buffer after the second pass – the reflected scene cleared outside the stencil

Reflection Example

The color buffer after the final pass

Reflected Scene First (issues)

- If the mirror is infinite, there is no need for the second pass
  - But might want to apply a texture to roughen the reflection
- If the mirror plane is covered in something (a wall) then no need to use the stencil or clear the color buffer in pass 2
- Objects behind the mirror cause problems:
  - Will appear in reflected view in front of mirror
  - Solution is to use clipping plane to cut away things on wrong side of mirror
- Curved mirrors by reflecting vertices differently
- Doesn’t do:
  - Reflections of mirrors in mirrors (recursive reflections)
  - Multiple mirrors in one scene (that aren’t seen in each other)

Rendering Normal First

- First pass:
  - Render the scene without the mirror
- Second pass:
  - Clear the stencil, Render the mirror, setting the stencil if the depth test passes
- Third pass:
  - Clear the depth buffer with the stencil active, passing things inside the mirror only
  - Reflect the world and draw using the stencil test. Only things seen in the mirror will be drawn

Normal First Addendum

- Same problem with objects behind mirror
  - Same solution
- Can manage multiple mirrors
  - Render normal view, then do other passes for each mirror
  - Only works for non-overlapping mirrors (in view)
  - But, could be extended with more tests and passes
- A recursive formulation exists for mirrors that see other mirrors