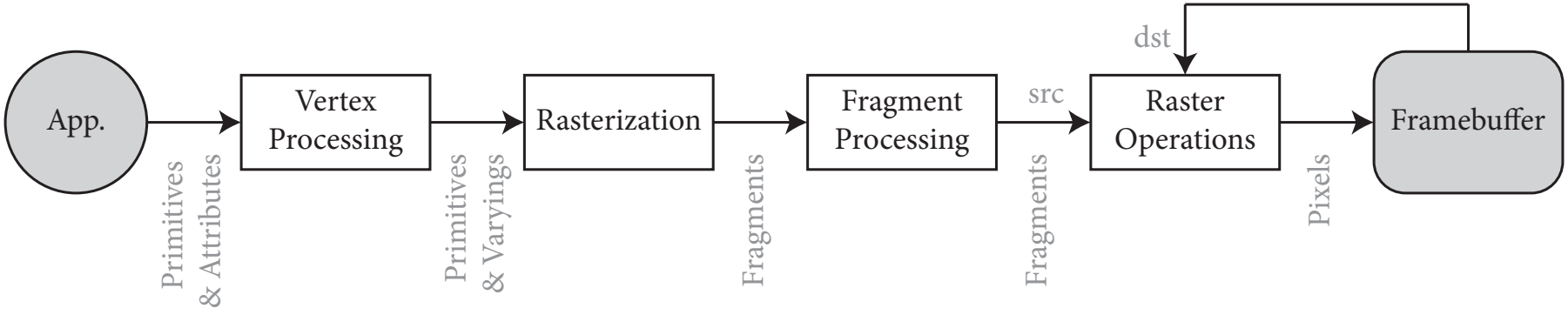
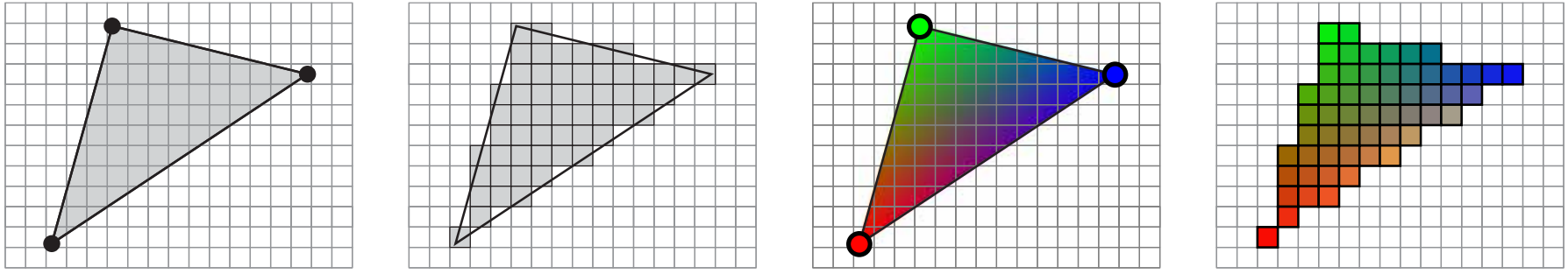
CSC 4356 / ME 4573 Interactive Computer Graphics

Basic Pipeline: Rasterization

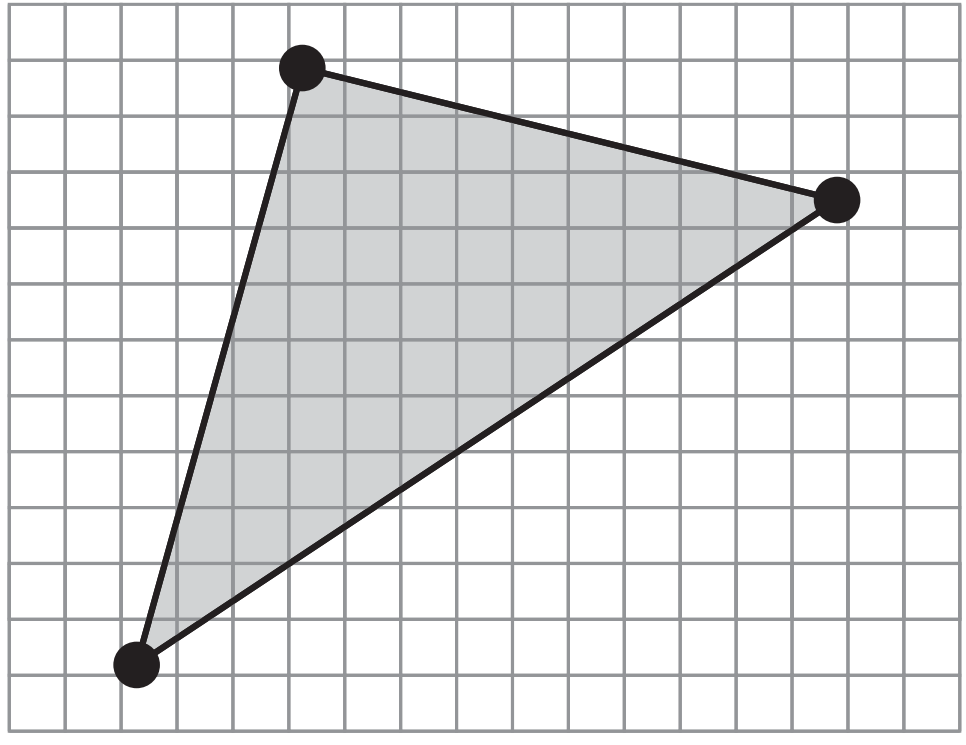
The Basic 3D Graphics Pipeline



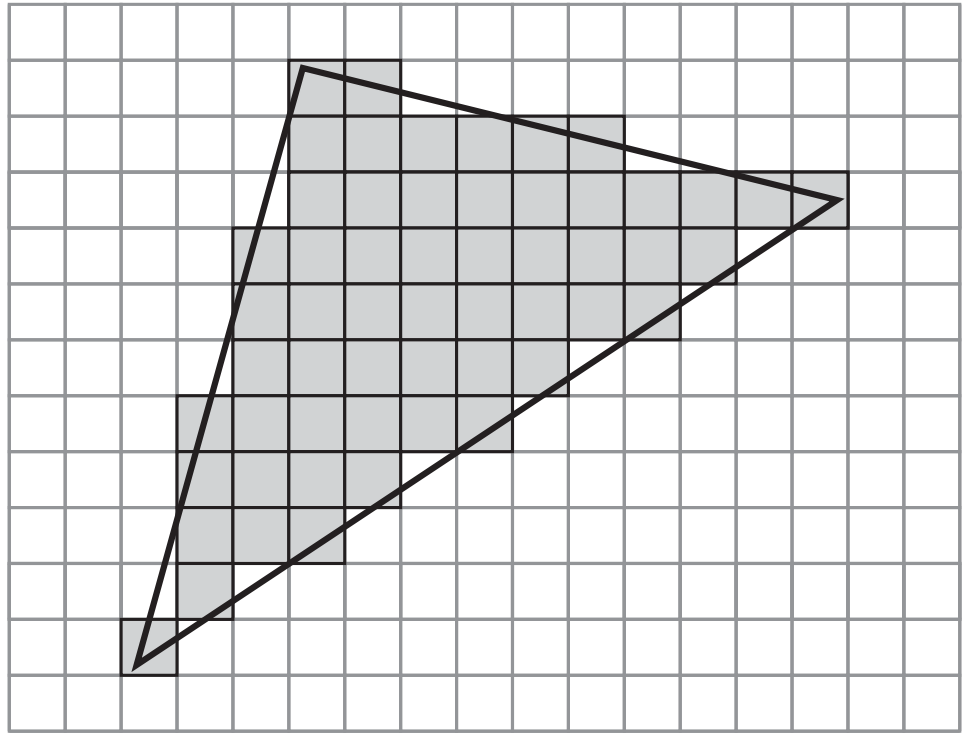
Rasterization



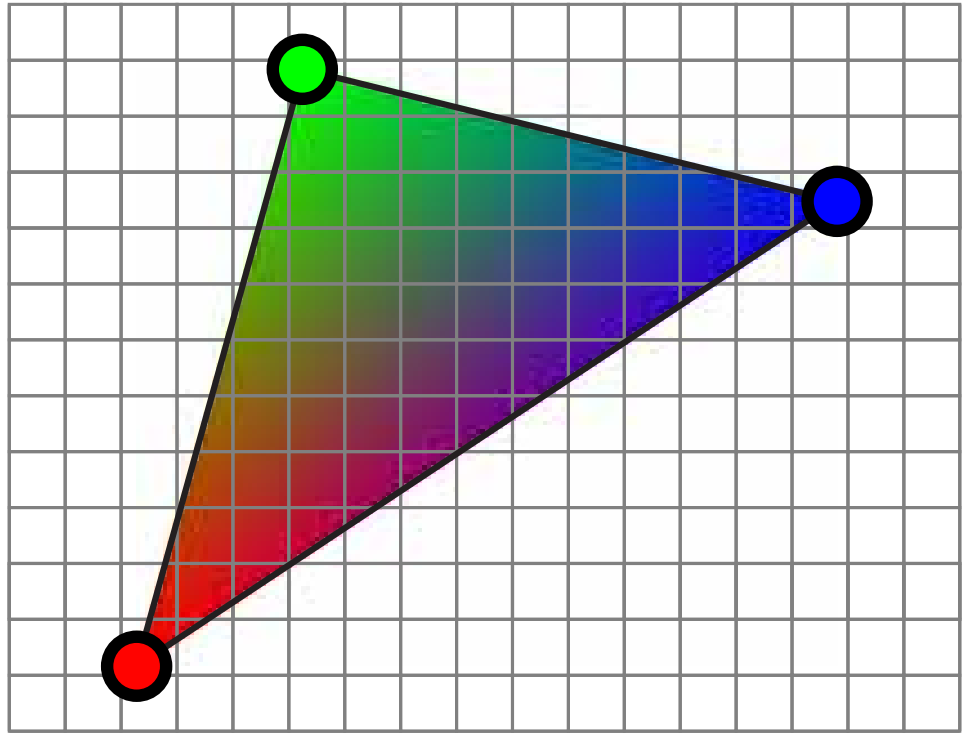
The process by which *primitives* become *fragments*.



Vertices are transformed into *screen space*.

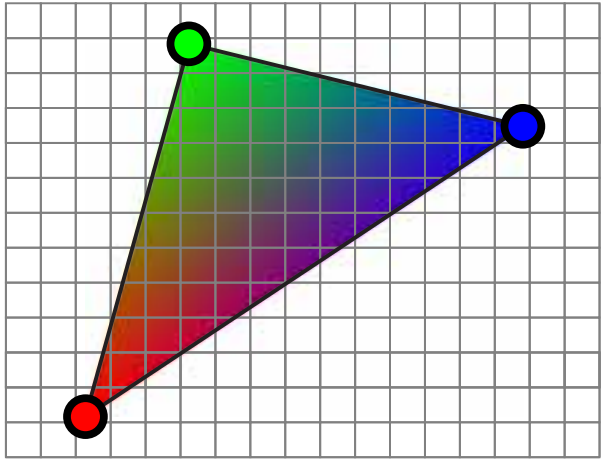


On-screen pixels lying within the triangle are processed.



A vertex has several attributes in addition to its position.

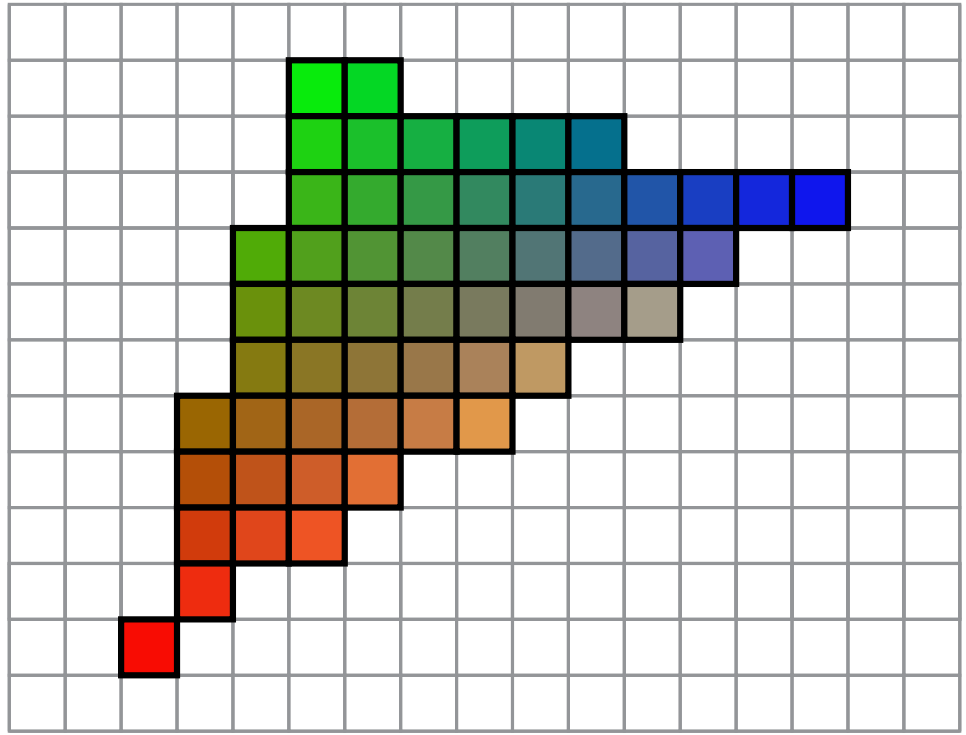
Varying Attributes



Attributes output from the Vertex Processing stage include:

- The colors resulting from lighting calculations
- Texture coordinates *
- Fog coordinates *

* Things we'll worry about later.



These attributes are *interpolated* across the triangle.
They are the input to the Fragment Processing stage.

Fragment Processing

The Fragment stage accepts the *varying* values from the rasterizer.

It uses them to determine the final color of the fragment. This may include:

- Texture reference *
- Fog calculation *

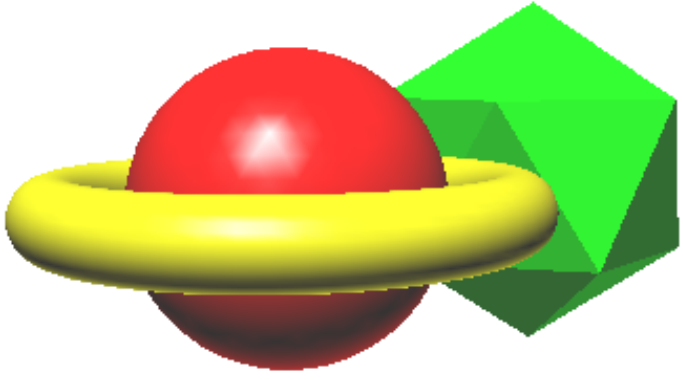
* More things we'll worry about later.

Raster Operations

Raster Operations merge the generated fragment with the pixel already found in the frame buffer.

- Depth Test
- Blending

Depth Test



- Rendering is like painting.
- A second brush-stroke covers the first.
- So too, a second rendered object obscures the first, even if it's farther from the viewer.

The *depth test* resolves this using a *depth* buffer.

9	9	9	9	9
9	9	9	9	9
9	9	9	9	9
9	9	9	9	9
9	9	9	9	9

Begin rendering with a color buffer cleared to white,
and a depth buffer cleared to “far.”

Rendering order no longer matters.

9	9	9	9	9
6	6	6	6	5
6	6	6	5	5
1	1	1	2	2
1	1	1	1	2

Begin arbitrarily with the yellow torus. The area appears yellow. Near and far areas are apparent in the depth values.

The sphere's depth is less than the far side of the torus, so red is written to the color buffer.

4	9	9	9	9
3	4	6	6	5
3	3	6	5	5
1	1	1	2	2
1	1	1	1	2

The sphere's depth is greater than the near side of the torus, so the yellow remains.

Finally, render the green icosahedron.

4	8	8	8	8
3	4	6	6	5
3	3	6	5	5
1	1	1	2	2
1	1	1	1	2

Both the sphere and the torus obscure it.

Relevant OpenGL

- `glutInitDisplayMode(GLUT_DEPTH);`
- `glClear(GL_DEPTH_BUFFER_BIT);`
- `glEnable(GL_DEPTH_TEST);`

Depth Test is used in almost every OpenGL application, including our simple example.

Blending

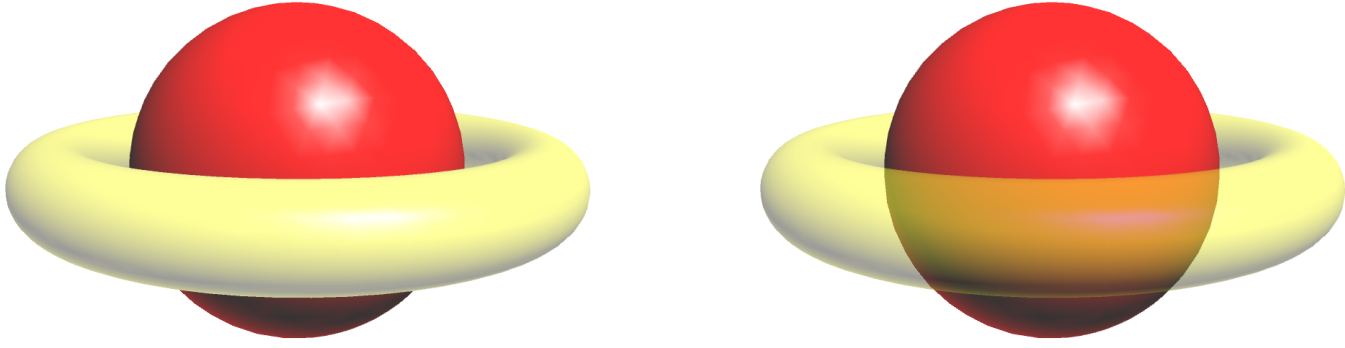
The pixel need not be simply *written* to the color buffer, it may be *blended* with it.

The *alpha* channel of an RGBA image gives a per-pixel mixing parameter.

The result is *transparency*.

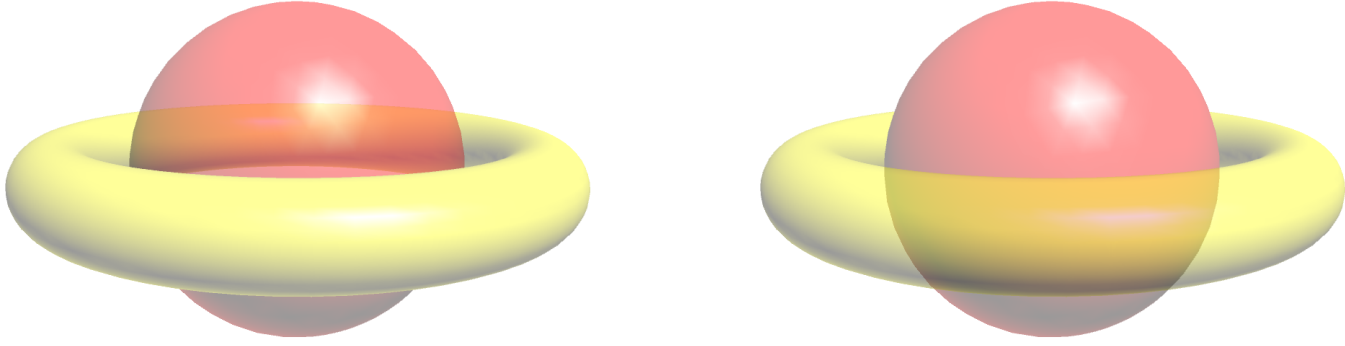
This comes with many difficult issues.

Suddenly, rendering order matters again.
On the left, yellow–red. On the right, red–yellow.



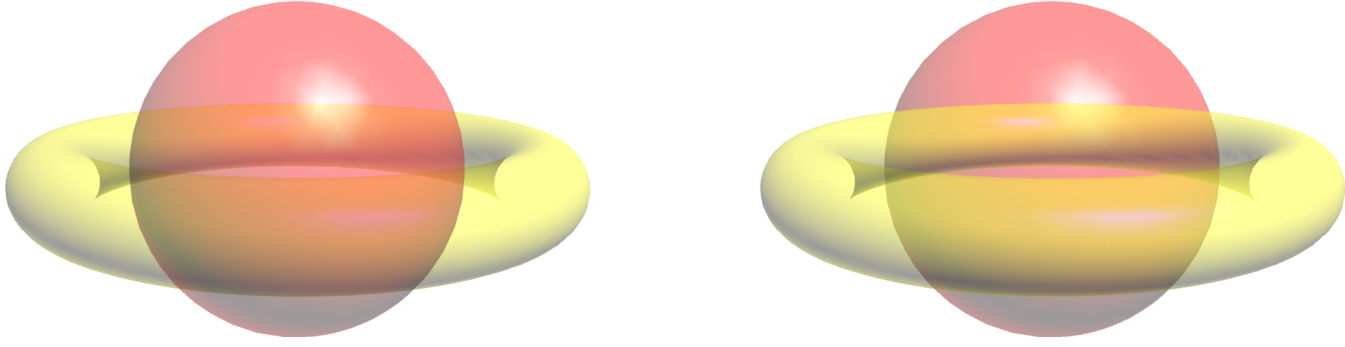
The yellow torus is transparent.
Transparent objects must be rendered *after* opaque.

When looking through *two* transparent objects,
there is *no* general solution.



The depth test occludes geometry that should be visible,
regardless of rendering order.

Depth buffer writes should be *disabled*
when rendering transparency



This allows all transparent geometry to remain visible,
but can undermine front-back relationships.

Relevant OpenGL

- `glEnable(GL_BLENDING);`
- `glBlendFunc(GL_ONE, GL_ONE);`
- `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);`
- `glDepthMask(GL_FALSE);`
- `glDepthMask(GL_TRUE);`

Coming up…

We'll spend the next few lectures on
vertex processing: transformation, and viewing.