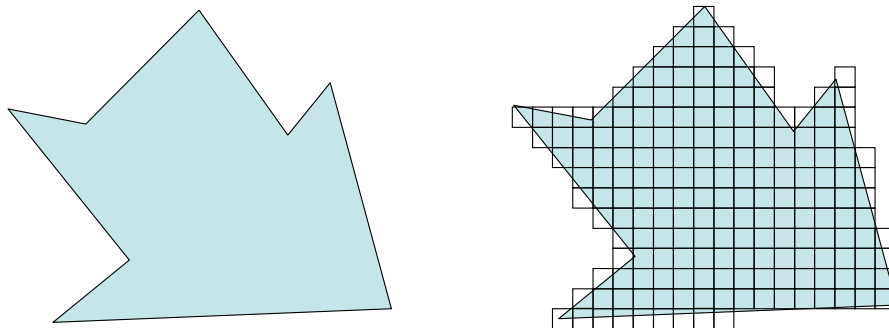


Polygons and Shading

- The last major step in the journey to the viewport is the actual painting of the polygons
- Painting the polygon includes a number of substeps:
 - How does the polygon translate into its final pixels?
 - Which pixels of the polygon are painted (with z -buffer HSR, painting and HSR occur in the same loop)?
 - What color(s) should be used?
 - Color based on absolute color values or lighting model?
 - Single color for the entire polygon or shaded?
 - Blended? Mapped from texture(s)?

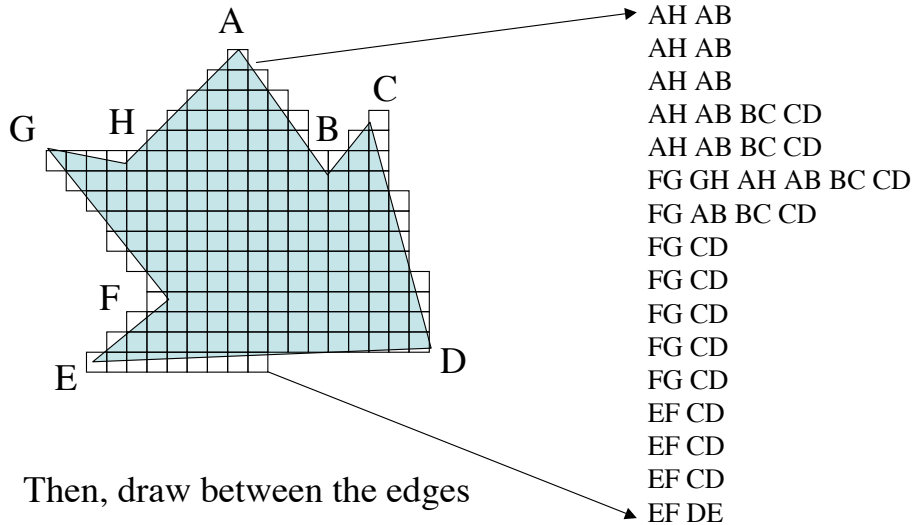
Scan-Line Conversion

- Converting the floating-point vertices of the polygon into the final set of discrete pixels
- We do this over and over, so it must be as fast as possible



Scan-Line Conversion Algorithm

- From top to bottom, build an *active edge table* showing what edges are active per scan line

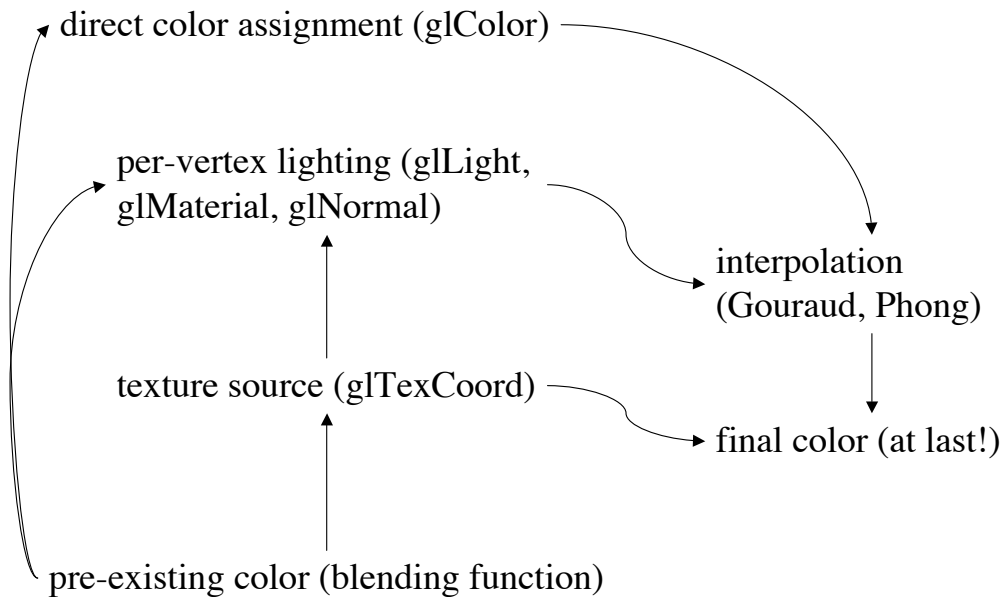


- Then, draw between the edges

Now, For Each Pixel...

- As we go down the scan lines, and move horizontally across the pixels, additional things happen:
 - Compare the z value at that pixel to the current z in the depth buffer — if the current z is closer, then we skip this pixel: it is occluded
 - If we decide to draw this pixel, then we record its z value in the depth buffer, then decide on a color
- Colors can be:
 - Assigned per vertex — in which case we need to determine what color to use *in between* vertices
 - Assigned per vertex as a material, which means that we have to factor in the lights to determine the final color — note how this is still per vertex, so the question of how to light the pixels in between vertices remains
 - Mapped to the entire polygon from a texture — so this is no longer per vertex, taking care of the entire polygon instead
 - A combination of all of these, including whatever color might already be there (i.e. blending)

The Color Chain



Color from Lighting

- Light model approximates physics, but only approximates
- In general:
 - Light sources add up (i.e. red light combines with blue light to produce magenta light)
 - Light intensity are modified by:
 - Angle of incidence for diffuse light (the closer to the negative of the normal, the brighter)
 - Distance from polygon for attenuated light
 - Shininess of material for specular components
 - Light and material are multiplied (i.e. when magenta [1.0, 0.0, 1.0] hits yellow [1.0, 1.0, 0.0], you get [1.0 * 1.0, 0.0 * 1.0, 1.0 * 0.0] = red)
 - Components (ambient, diffuse, specular) are added up to get the final color — per vertex
- Note how this model is solely between a polygon and its light sources — no interactions with other objects. Thus, no reflections and no shadows. If we want this, we need to do more.

Color from Texture

- Texture coordinate-to-vector mapping determines how a texture is “wrapped” onto a polygon
- Intervening pixels are approximated using ratios
- The mapped texture color can be treated in a number of ways:
 - Like the material of the object at that pixel
 - Like a color to be blended by some function
- If textures *and* lighting are active, the mapped texture color is combined with the current light color using some customizable function

Pre-Existing Color/Blending

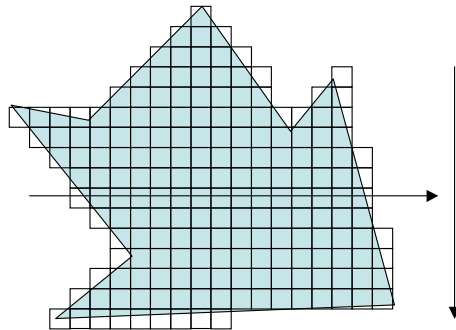
- When blending, the current pixel is sampled to see what color is already there
- This color is combined with the calculated color using some function — lots of ways to do this
- The final “blended” color is the result of this function
- Note how blending is sensitive to the order in which you draw your model — also it must not perform HSR, or else occluded pixels will never make it to the frame buffer!

Interpolation/Shading

- Note that if we get this far, in many cases of the color chain, we only have the colors *at the vertices*
- How do we determine the color(s) in between?
- Two general algorithms: Gouraud and Phong
- Gouraud shading is the less computationally expensive algorithm; it is what OpenGL uses
- Shading has so many variations and possibilities that OpenGL 2.0 has introduced *programmable shaders*

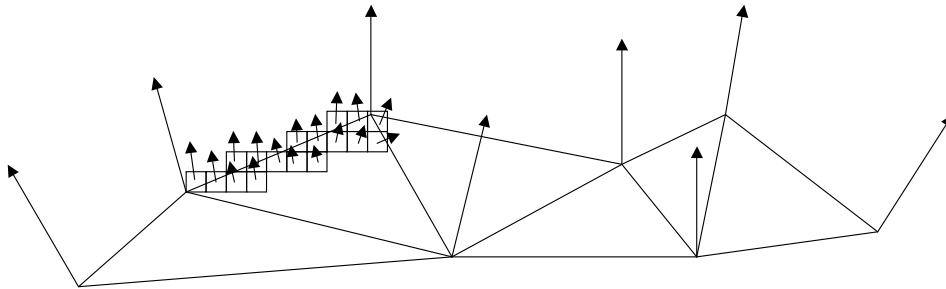
Gouraud Shading

- Take the colors at the vertices, then interpolate vertically
- Within each scan line, interpolate horizontally



Phong Shading

- Used mainly in conjunction with lighting, because it assumes that the normal at a pixel influences the final color
- Instead of calculating the color for each vertex then interpolating the color, Phong shading *interpolates the normal* at each pixel and performs the lighting calculation *per pixel*
- Overall shading quality is better but of course it's more work



Even More Possibilities!

- Note that despite all of these phases and algorithms, we still aren't close to having a completely realistic rendering
- As mentioned, arbitrary/generalized reflections and shadows are still missing
- Other real-world features:
 - Bump mapping — textures that affect the 3D surface of a polygon (tree bark, dimples on a golf ball)
 - Filaments/fibers (fur, hair, cloth)
 - Liquids/refraction
- Keep an eye on the latest Pixar movie or id game for the state of the art...