

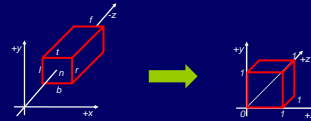
ECE 595 / CS 491 / CS 591  
**Real-Time Rendering &  
 Graphics Hardware**

Pradeep Sen  
 Advanced Graphics Lab

Class 3  
 January 24, 2007

**Pop Quiz!**

- Write the orthographic projection matrix that projects the volume defined by the  $r$ ,  $l$ ,  $t$ ,  $b$ ,  $n$ ,  $f$  planes to the unit cube  $(0, 1)$
- $n$ ,  $f$  are defined positive and make sure the resulting projection puts view direction at  $+z$ .



Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

**Last time**

- Projection transformations
- Clipping

Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

**Today**

- Finish the rendering pipeline
  - Rasterization
  - Texturing
  - Framebuffer operations
- Discuss the project

Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

**Rasterization**

- There are two problems the rasterizer must solve:
  - Determine which pixels are "touched" by the triangle
  - Determine the value of interpolants at each of these vertices

Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

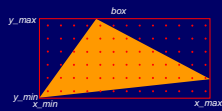
**Rasterization**

- Conversion from floating point screen space coordinates into integer pixel coordinates on the screen

Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

**Bounding box algorithm**

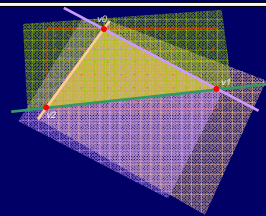


```
void rasterizeTriangle(void) {
  for (y = y_min; y < y_max; y++) {
    for (x = x_min; x < x_max; x++) {
      if insideTriangle(x,y) {
        generateFragment(x,y);
      }
    }
  }
}
```

Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

**Inside tests**



Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

**Inside tests**

```
void lineEquation(float *v0, float *v1, float *f){
  crossProduct(v1, v0, f);
}

bool insideTriangle(int x,y) {
  lineEquation(v0, v1, &f0);
  lineEquation(v1, v2, &f1);
  lineEquation(v2, v0, &f2);

  Pt p(x,y,1);
  if ((dotProduct(p, f0) > 0) &&
      (dotProduct(p, f1) > 0) &&
      (dotProduct(p, f2) > 0) {
    // inside the triangle
    return true;
  }
  return false;
}
```

Real-time Rendering & Graphics Hardware  
 Pradeep Sen

Class 3 - January 24, 2007

### Inside tests

To accelerate things, you can make it incremental:

```

void rasterizeTriangle(void) {
    // compute line equations f0,f1,f2 as before
    Pt p(x_min, y_min, 1);
    e0 = dotProduct(p,f0);
    e1 = dotProduct(p,f1);
    e2 = dotProduct(p,f2);

    for (y = y_min; y < y_max; y++) {
        for (x = x_min; x < x_max; x++) {
            if (e0 >= 0 && (e1 >= 0) && (e2 >= 0) {
                generateFragment(x,y);
            }
            e0 += f0.i; e1 += f1.i; e2 += f2.i;
        }
        e0 = -(x_max - x_min)* f0.i + f0.j;
        e1 = -(x_max - x_min)* f1.i + f1.j;
        e2 = -(x_max - x_min)* f2.i + f2.j;
    }
}

```

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

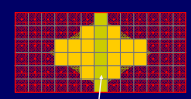
### Boundary cases

- We need to prevent double counting of pixels on triangle boundaries
- This is important for transparency, CSG, and efficiency
- Let's first look at a few wrong ways to handle it

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Incorrect solution #1

- Ignore the condition and double rasterize
- Imagine we are rendering semi-transparent triangles:

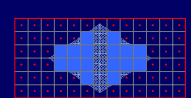


pixels rendered twice will be of a different color

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Incorrect solution #2

- Set the inside test to '>' to ignore samples that lie exactly on the line edge
- No two triangles will "touch" the same pixels unless they overlap, in which case you want to rasterize both anyway
- Problem:



AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Shadow test

- Modify our line side test...

```

bool insideLine(float e) {
    return (e >= 0);
}

bool insideLine(float e, float *f) {
    return (e == 0) ? shadowTest(f) : (e > 0);
}

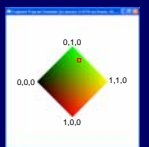
bool shadowTest(float *f) {
    return (f.i == 0) ? (f.j > 0) : (f.i > 0);
}

```

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### What about shading?

- So now we have a simple way to determine which pixels are on.
- Now how do we color the pixel?



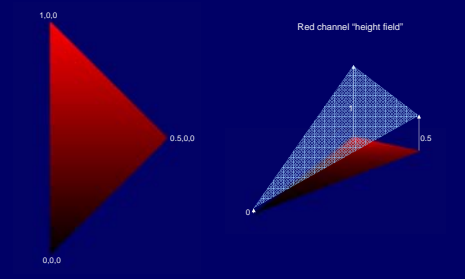
AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Interpolation

- Want to represent a function over the surface of the triangle
- Many ways to do this...
- Graphics hardware does it implicitly from the values at the corners of the triangle

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

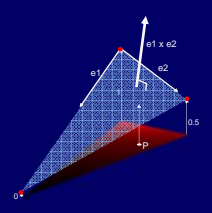
### Interpolation



AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

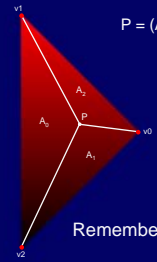
### How do you compute this plane?

- Equation of a plane



AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

## Barycentric coordinates



$$P = (A_0/A) v_0 + (A_1/A) v_1 + (A_2/A) v_2$$

$$P = \alpha_0 v_0 + \alpha_1 v_1 + \alpha_2 v_2$$

$$\alpha_0 + \alpha_1 + \alpha_2 = 1$$

$$(A_0/A) + (A_1/A) + (A_2/A) = 1$$

Remember: this is done in screen space!

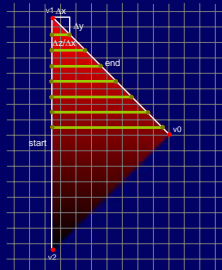
## Interpolants

- You can interpolate:
  - Colors (3)
  - Depth (1)
  - Normals (3)
  - Texture coordinates (1,2,3)
  - etc.

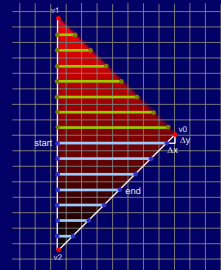
## Status

- So now we have an algorithm that goes through the bounding box and determines which pixels should be rasterized by the triangle
- Then we must work out the interpolated value separately for each pixel
- Very inefficient...

## Edge-walking rasterization



## Edge-walking rasterization



## Triangle rasterization algorithm

```
void rasterizeTriangle(void) {
    // sort vertices in order of descending y
    // identify scanlines of first half of the triangle
    // compute screen-space derivatives along edges

    for each scanline {
        // find span by interpolating vertex coordinates
        // find start and end values for interpolants
        // (color, depth, texture coordinates)
        // rasterize span

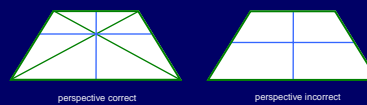
        for each pixel in span {
            // interpolate color, depth, texture coords
            // perform depth buffering
            // shade pixel
        }
    }

    // repeat for the second half of the triangle
}
```

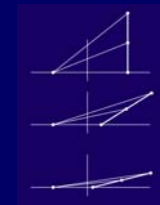
## Acceleration

- Digital Differential Analyzer (DDA)
  - Represent slope and accumulation with fixed point
  - Have to be careful to avoid overflow
- Homogeneous recursive descent
  - Used in modern GPU's

## Perspective-correct interpolation



## Perspective-correct interpolation



### Solution

- Project all interpolants first by dividing by  $w$
- Interpolate interpolants as well as  $1/w$
- Get original value back by multiplying by interpolated  $w$

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Texture mapping

standard geometry

texture-mapped geometry

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Essentially a lookup table

- You want to evaluate function  $f(s,t)$  on the surface...
- You can store it in an array for fast access

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Nearest neighbor sampling

- Map the point to the 2D array of samples
- Take the sample that is closest to the projected point

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Bilinear interpolation

- Linear reconstruction of the function  $f(s,t)$  using neighboring color samples

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Framebuffer operations

- Framebuffer is essentially a large memory array to store the color, depth, stencil data
- Initial framebuffers (circa 1975) were very small (e.g. Evans and Sutherland)
- Made possible by large arrays of DRAM

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Color buffer

- Essentially a large array of color values
- Read out for display to the screen
- Allows for blend mode operations, e.g.:
 
$$f = \alpha C_{src} + (1-\alpha) C_{dst}$$

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Depth buffer

- Solution to the hidden surface problem
- Generate depth for every fragment rasterized
- Compare to the value currently in the buffer, replace if smaller
- Essentially:
 

```
void depthBufferID(void) {
    z = Z_MAX;
    for each fragment rasterized f {
        if (f.z < z) {
            z = f.z;
            write f into framebuffer
        }
    }
}
```

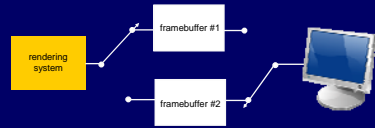
AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

### Stencil buffer

- Essentially a state counter at each pixel
- Can keep track of how many times each pixel was rasterized
- Can be used for various algorithms, such as stencil shadows

AGL Real-time Rendering & Graphics Hardware Pradeep Sen Class 3 - January 24, 2007

## Double buffering



## That's the pipeline!

- We started off with triangles and ended up with pixels on the screen!
- Now you get to implement it yourself

## Introductory project

- Due Feb 7

## Reading

- Olano and Greer paper