

ECE/CS 433 Introduction to Computer Graphics

Class 17
October 18, 2007

Pradeep Sen
Advanced Graphics Lab



Announcements

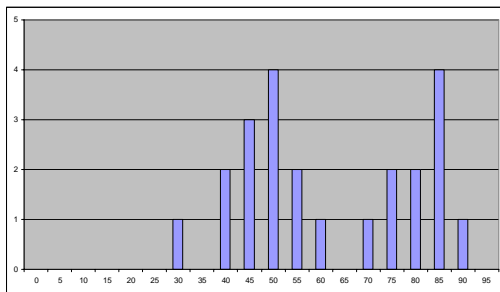
- Return the exam



ECE/CS 433 Introduction to Computer Graphics
Pradeep Sen

Class 17 – October 18, 2007

Exam grade distributions



ECE/CS 433 Introduction to Computer Graphics
Pradeep Sen

Class 17 – October 18, 2007

“Immersive Experiential Content: Hologram to Holodeck”



Pete Rogina
Worldscape

Friday, October 19, 2007
ME 208 at noon



Announcements

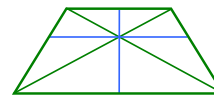
- Return the exam
- GFX Café
- After class, short Q/A for hw3



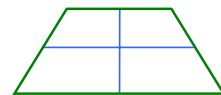
ECE/CS 433 Introduction to Computer Graphics
Pradeep Sen

Class 17 – October 18, 2007

Perspective-correct interpolation



perspective correct



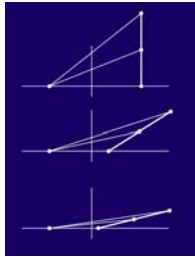
perspective incorrect



ECE/CS 433 Introduction to Computer Graphics
Pradeep Sen

Class 17 – October 18, 2007

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

Once fragments are rasterized

- They can be shaded (colored)
- Texture mapping
- Fragment program
- Etc.

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

Color buffer

- Essentially a large array of color values
- Read out for display to the screen
- Allows for blend mode operations, e.g.:

$$f = \alpha C_{\text{src}} + (1-\alpha) C_{\text{dst}}$$

Depth buffer

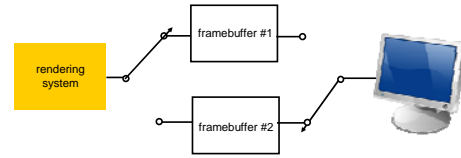
- Solution to the hidden surface problem
- Generate depth for every fragment rasterized
- Compare to the value currently in the buffer, replace if smaller
- Essentially:

```
void depthBuffer1D(void) {  
    z = Z_MAX;  
    for each fragment rasterized f {  
        if (f.z < z) {  
            z = f.z;  
            write f into framebuffer  
        }  
    }  
}
```

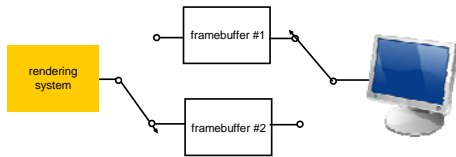
Stencil buffer

- 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

Double buffering



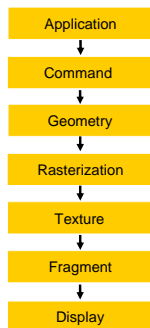
Double buffering



That's the pipeline!

- We started off with triangles and ended up with pixels on the screen!

Graphics pipeline

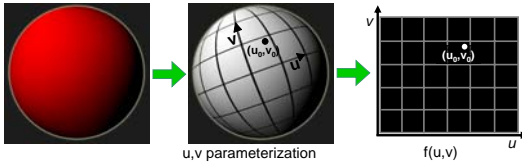


Let's look at a few parts in more detail...

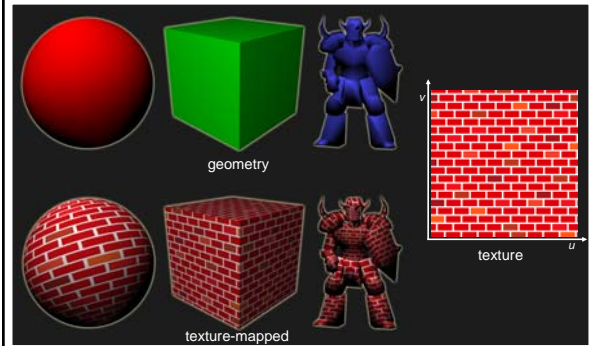
- Texture mapping

Introduction to texture mapping

- Developed by Catmull ('74) as way to improve realism without increasing geometric complexity
- Define parameterization (u,v) over surface
- Function f(u,v) can now modify appearance of surface
- Function f(u,v) can be stored as an array of pixels called a *texture*



Introduction to texture mapping



Representing Theoretical Texture Signal

There are several possible ways to represent the theoretical texture signal:

- Explicitly, with a mathematical equation

e.g. $f(u,v) = \sin(10\pi u v)$ →

Unfortunately, it is difficult to generalize this approach...

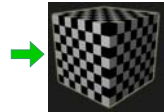


Representing Theoretical Texture Signal

There are several possible ways to represent the theoretical texture signal:

- Explicitly, with a mathematical equation
- Implicitly, with a functional program that can compute f(u,v) everywhere on the domain

```
Shader checkboard(float4 uv) {
    uv = floor(uv * 8);
    return ((uv.x % 2) == (uv.y % 2));
}
```



Representing Theoretical Texture Signal

There are several possible ways to represent the theoretical texture signal:

- Explicitly, with a mathematical equation
- Implicitly, with a functional program that can compute f(u,v) everywhere on the domain
- Implicitly, with a composition of geometrical shapes with well-defined mathematical representations.



Representing Theoretical Texture Signal

There are several possible ways to represent the theoretical texture signal:

- Explicitly, with a mathematical equation
- Implicitly, with a functional program that can compute f(u,v) everywhere on the domain
- Implicitly, with a composition of geometrical shapes with well-defined mathematical representations.
- Approximately, using a discrete representation



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
- But what happens to the screen samples that do not map directly to the samples of the texture?

Nearest neighbor sampling

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

Bilinear interpolation

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

Reading

- Angel thru Ch 8