

**ECE/CS 412**

# **Introduction to Computer Graphics**

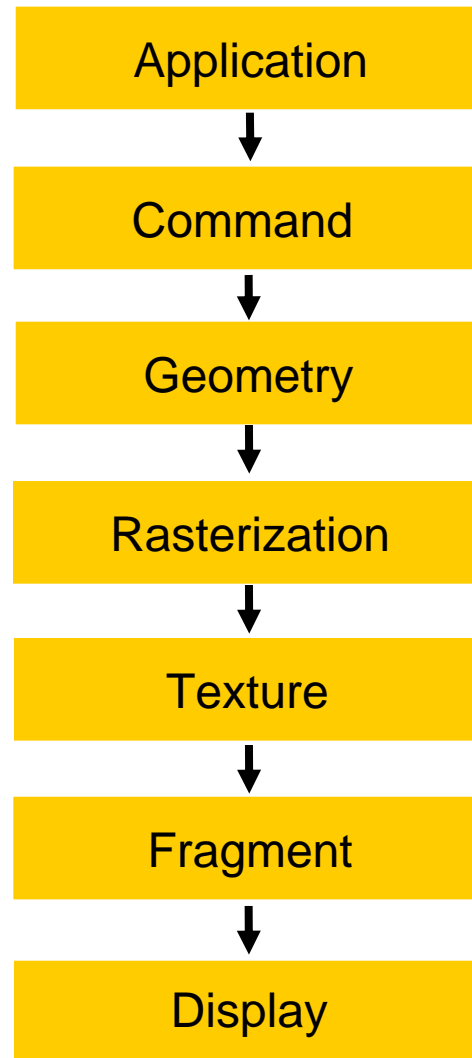
Class 15

Pradeep Sen  
Advanced Graphics Lab



# Graphics pipeline

---



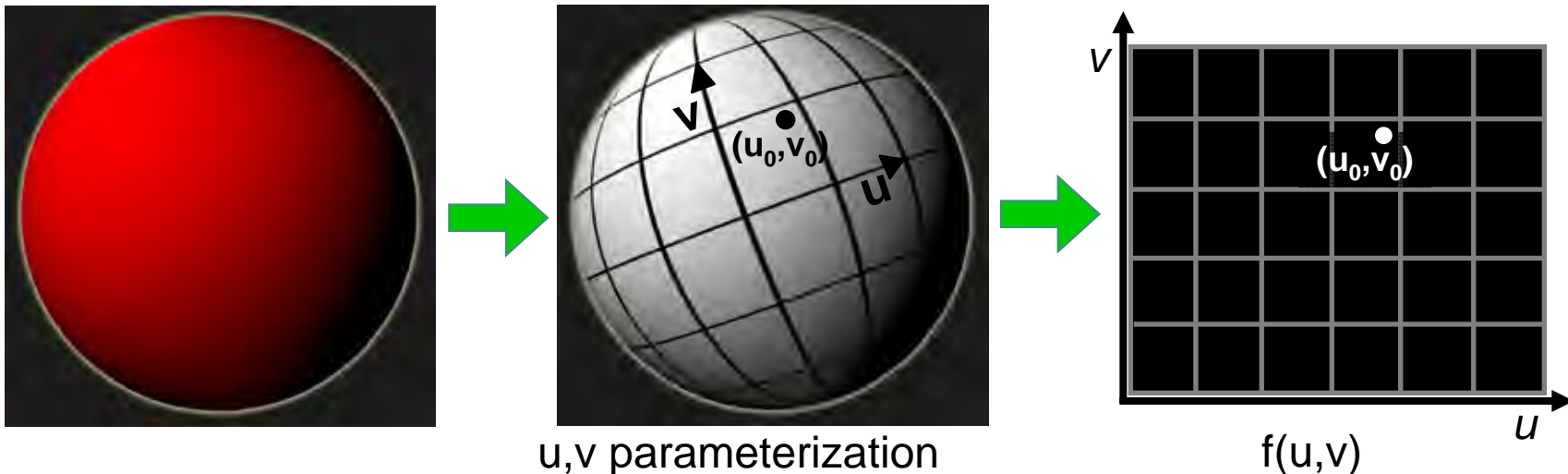
# Let's look at a few parts in more detail...

---

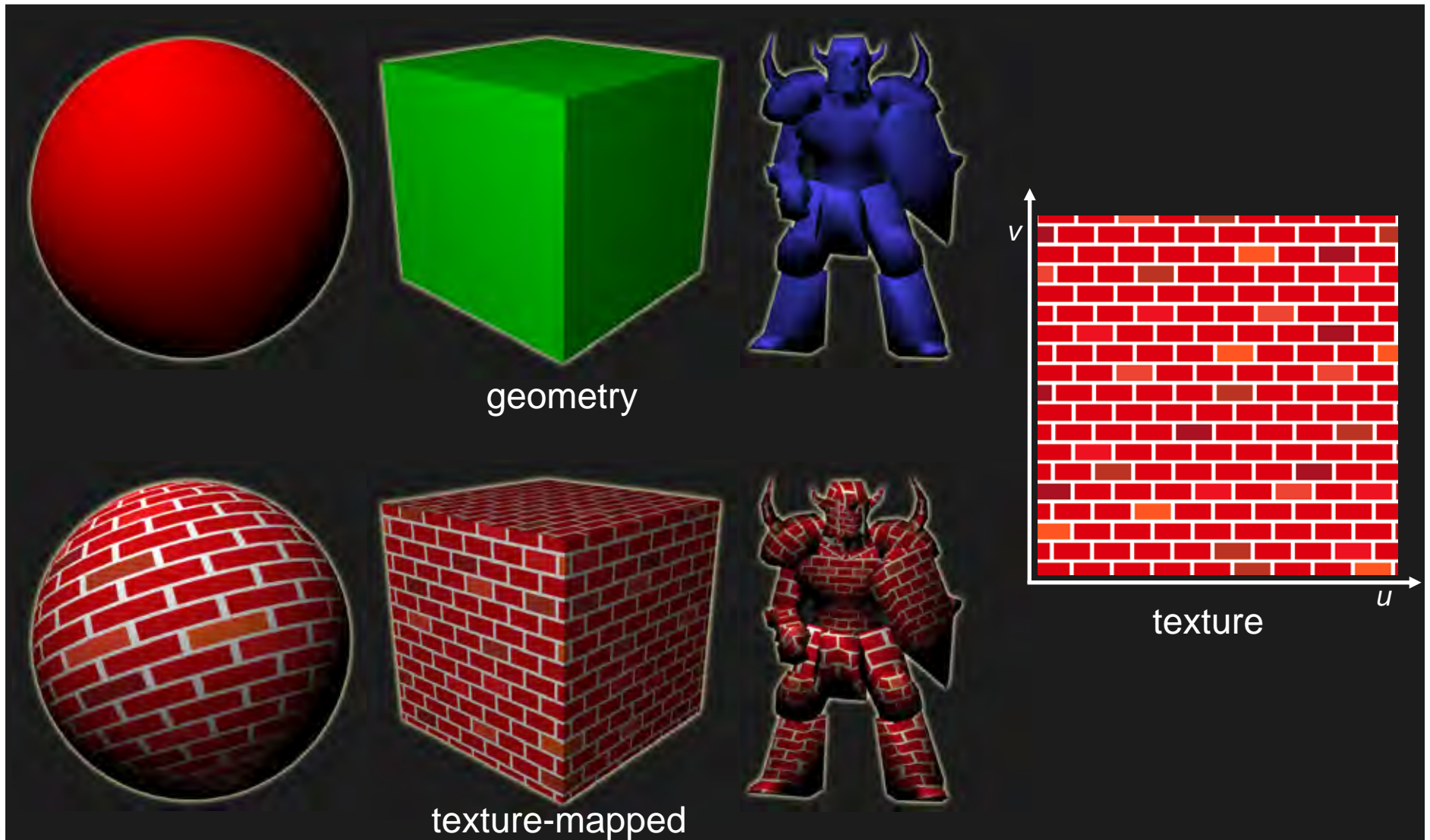
- Texture mapping

# 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



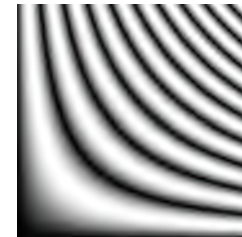
# Representing Theoretical Texture Signal

---

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...



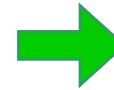
# Representing Theoretical Texture Signal

---

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.

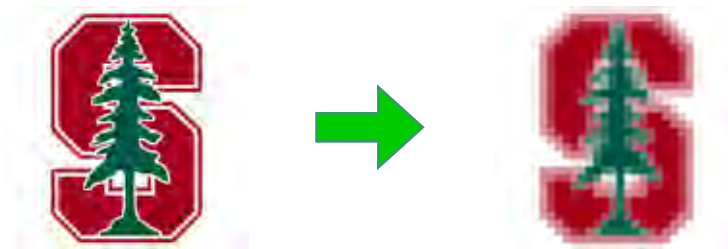
e.g. vector graphics

# 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

# Texture filtering

---

- Screen space samples are closer together than texture samples (magnification)
  - OpenGL provides two reconstruction filters: `GL_NEAREST`, `GL_LINEAR`
- Screen space samples are farther apart than texture samples (minification)
  - Summed area tables
  - Mipmapping

# Summed area tables

---



# Mipmapping

---

- MIP- *multum in parvo* (many things in a small place)
- Technique in which to perform pre-computed filtering for minification



# Mipmapping

---

- Accessing the mipmap



# Problems with mipmapping

---

- Decimated textures all have a square footprint (isotropic)



# Anisotropic filtering

---

- Approximate the pixel “footprint” with an ellipse
- Compute contribution in ellipse through multi-sampling

# Reading

---

- Angel thru Ch 8

