

ECE/CS 433

Introduction to Computer Graphics

Class 8

Pradeep Sen
Advanced Graphics Lab



Announcements

- HW 3 on the website
- Due Tuesday, September 29
- Don't wait until the last minute
- Submit through WebCT



Last time

- 2-D graphics and transformations



Today

- Finish up with 2-D transformations talk about the 3-D pipeline



2-D transformations

- We need a way to move objects around the screen

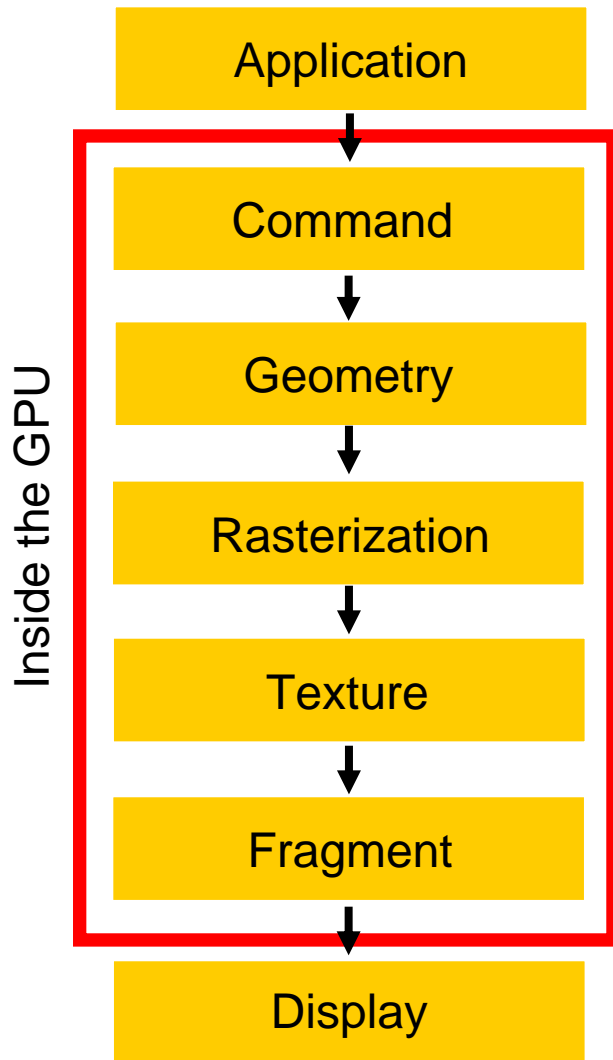


On to 3-D graphics!



ECE/CS 412 Introduction to Computer Graphics
Pradeep Sen

The real-time rendering pipeline

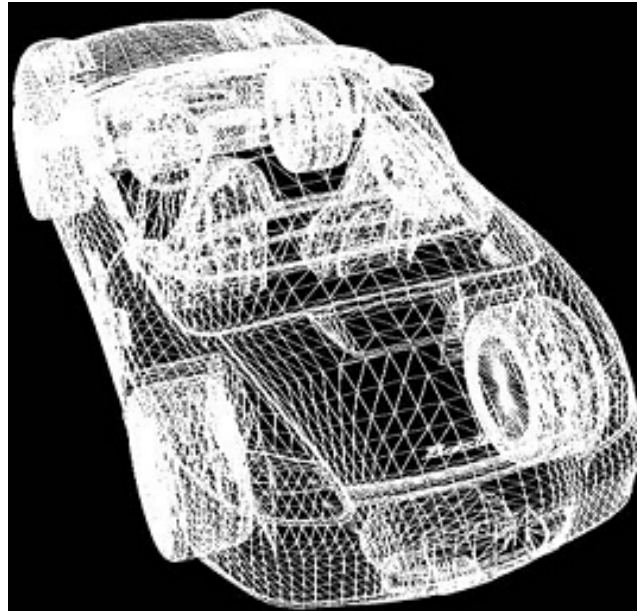


The application:

- Runs the game (keeps the state)
- Controls the enemy AI
- Handles user input
- Determines where the user is and what they are looking at
- Sets the state of the graphics hardware before drawing
- Once it is ready to draw, it sends the triangles to the graphics card for rendering

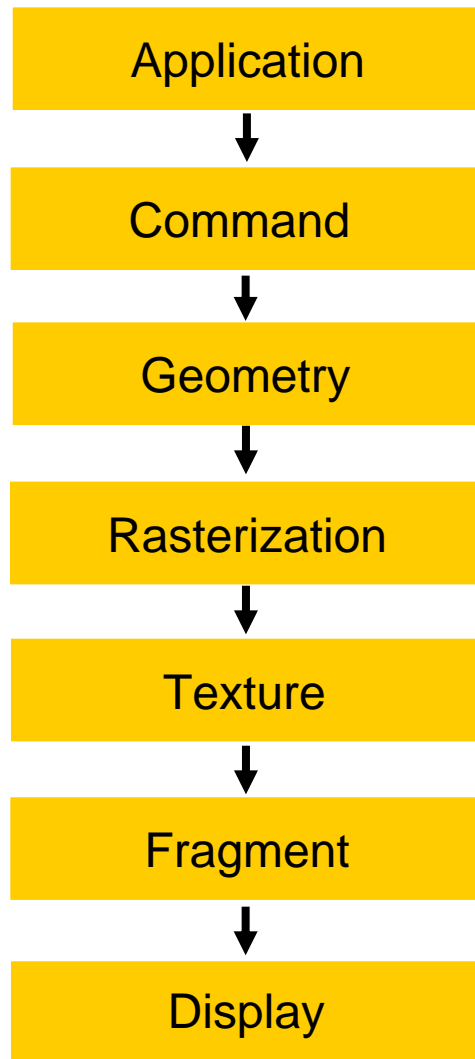
Triangles are king

- In real-time graphics, geometry is typically represented by triangles



- The GPU must draw these triangles to the screen

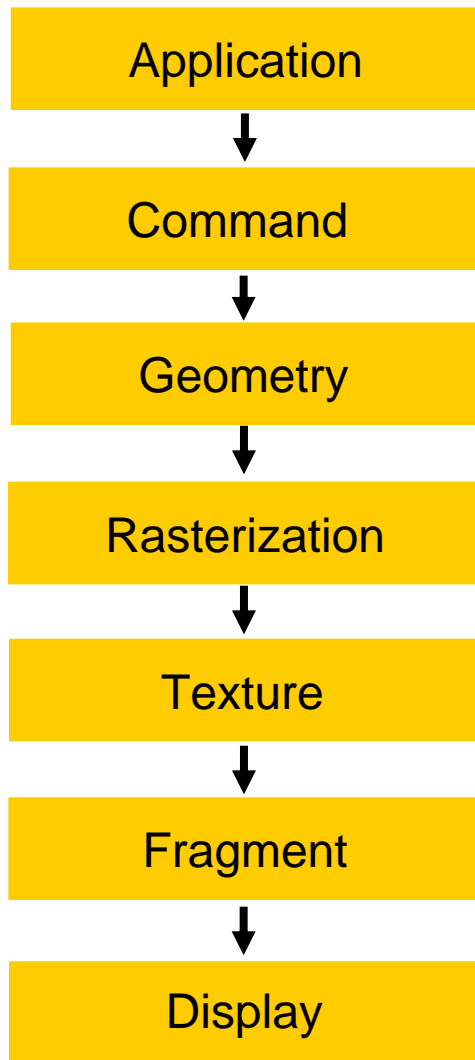
The real-time rendering pipeline



Command processing:

- Parses the input from the CPU
- Sets up the hardware for rendering

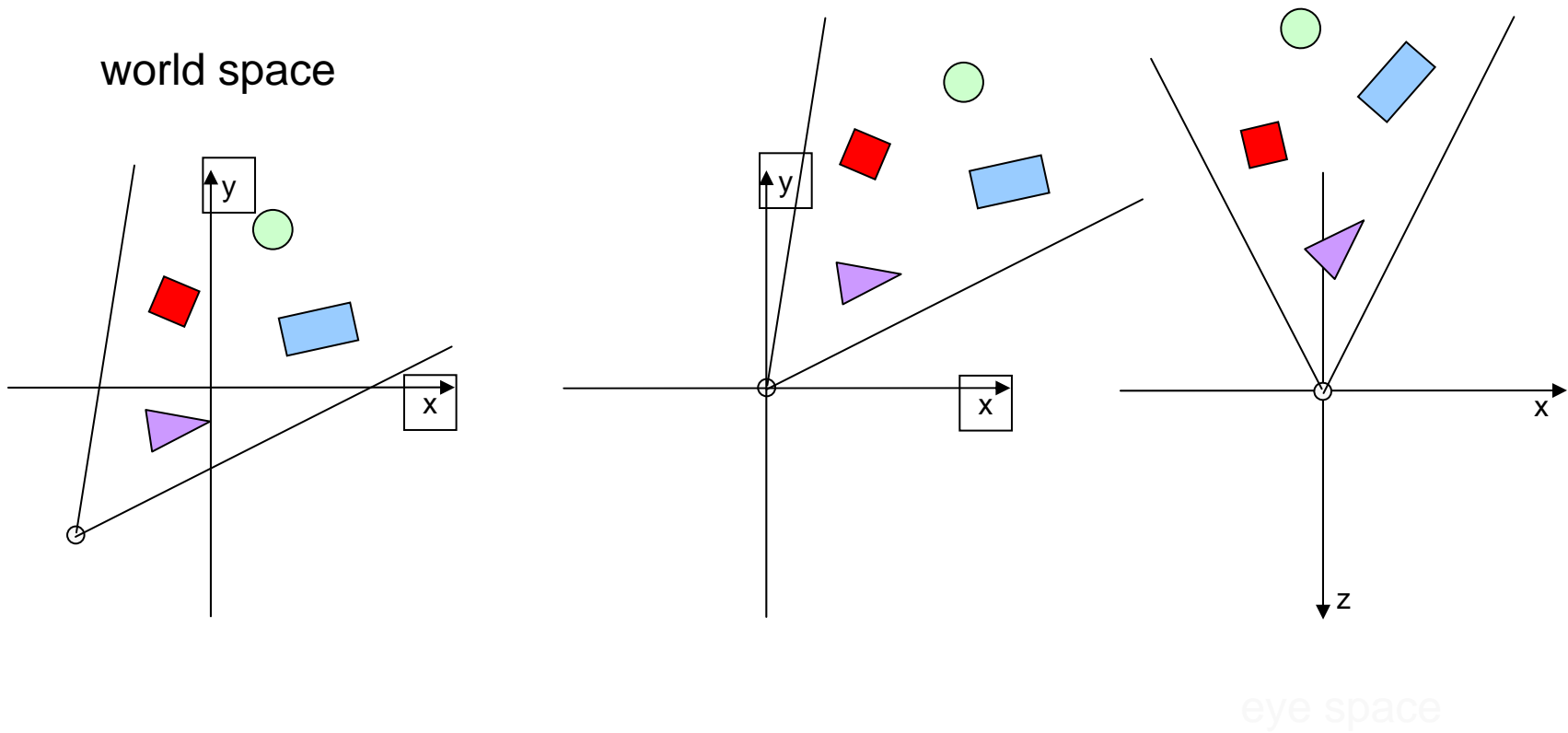
The real-time rendering pipeline



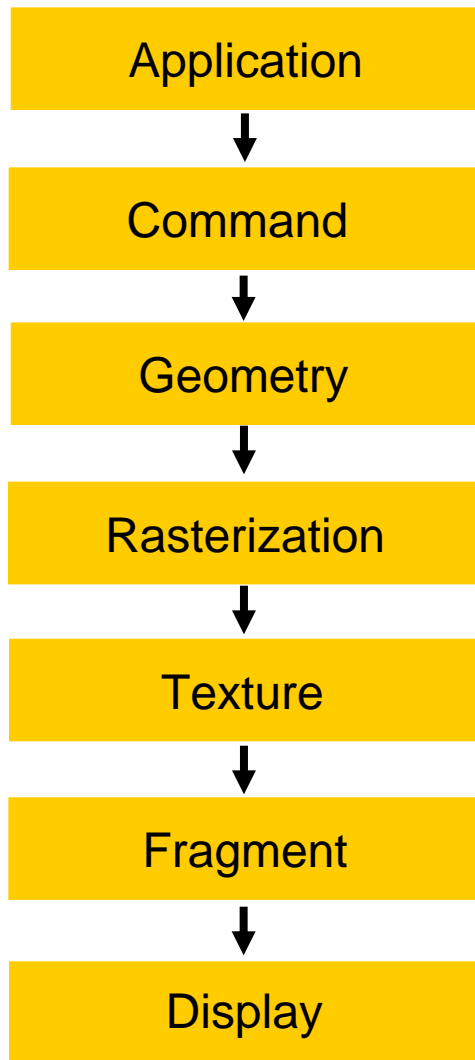
Geometry stage:

- Vertices are transformed
- Lighting can be computed
- Transformed vertices are projected to the screen
- Triangles are clipped
- Transformed to screen coordinates in preparation for rasterization

View Transform



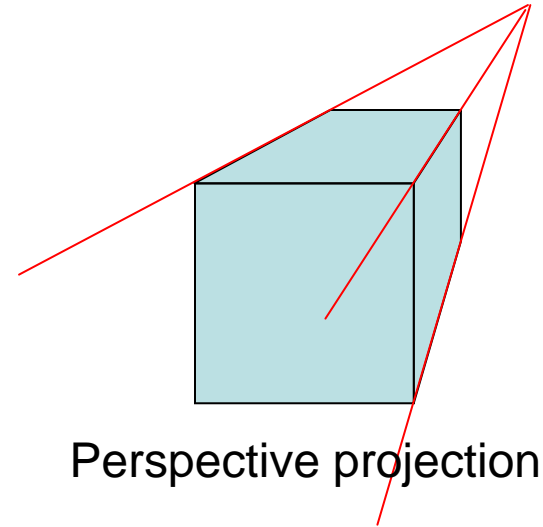
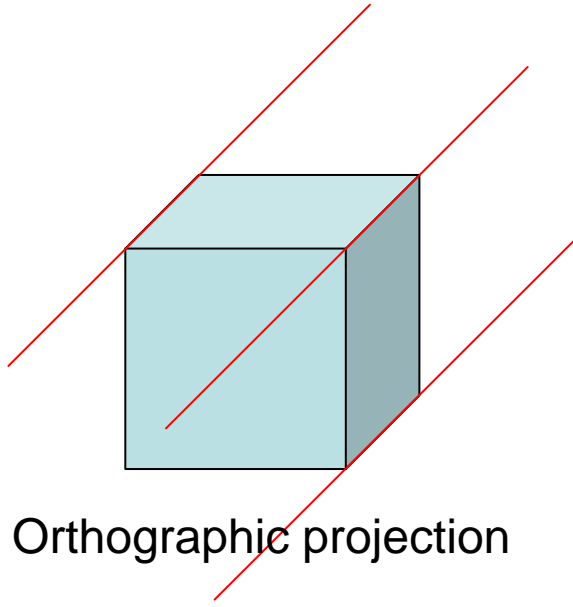
The real-time rendering pipeline



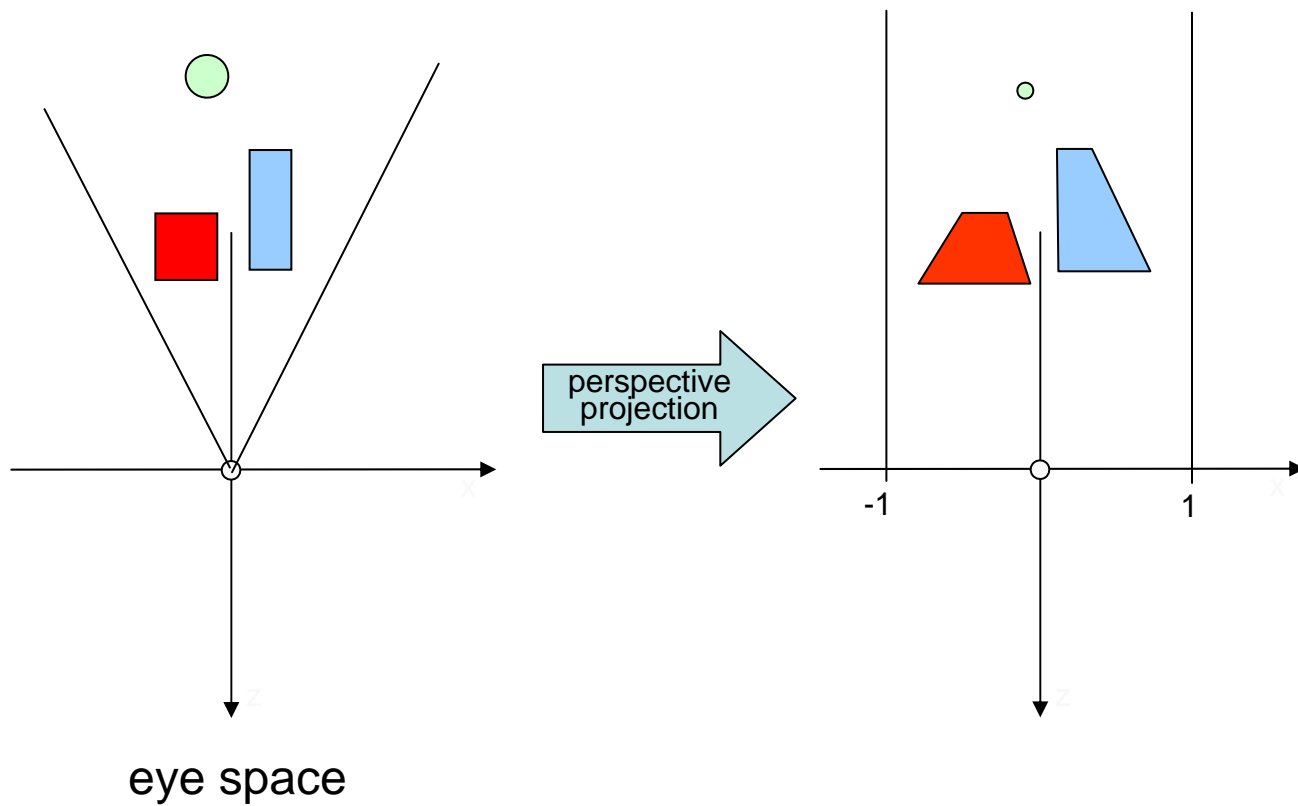
Geometry stage:

- Vertices are transformed
- Lighting can be computed
- Transformed vertices are projected to the screen
- Triangles are clipped
- Transformed to screen coordinates in preparation for rasterization

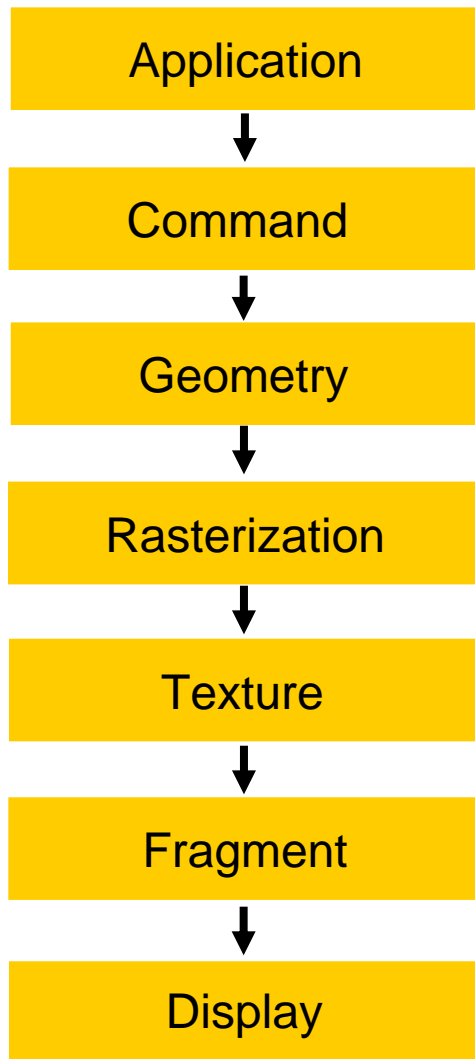
Kinds of projection



Perspective projection



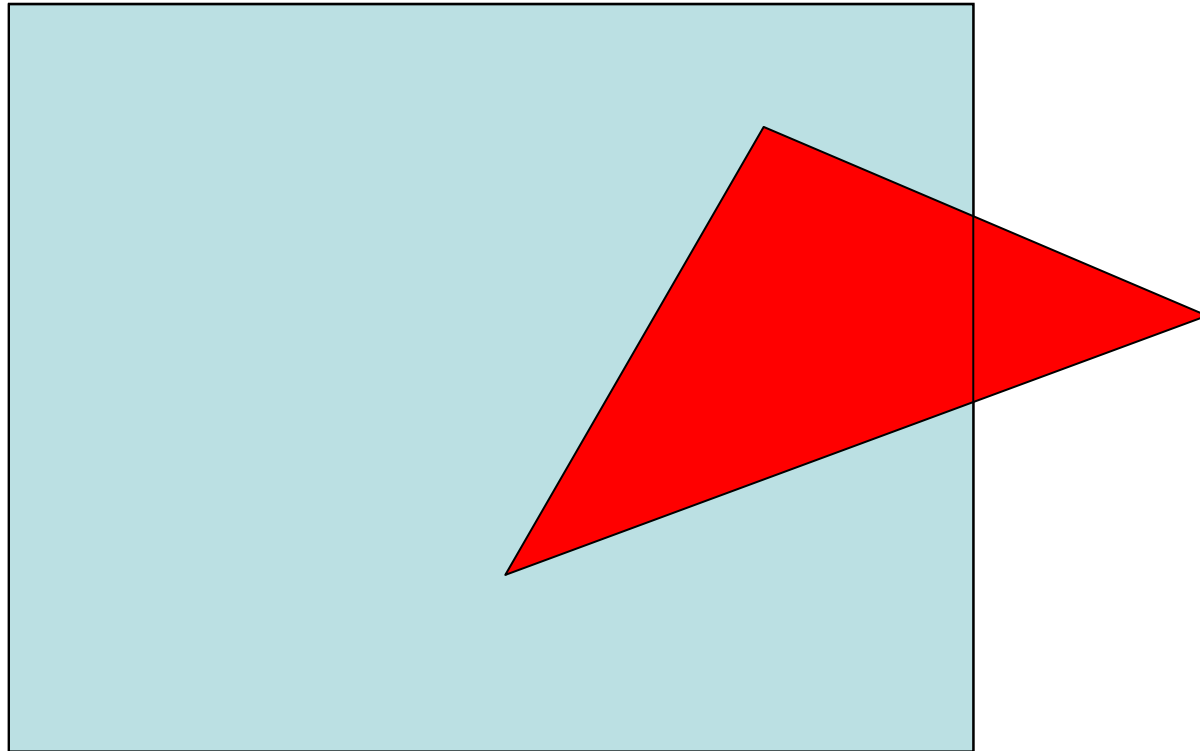
The real-time rendering pipeline



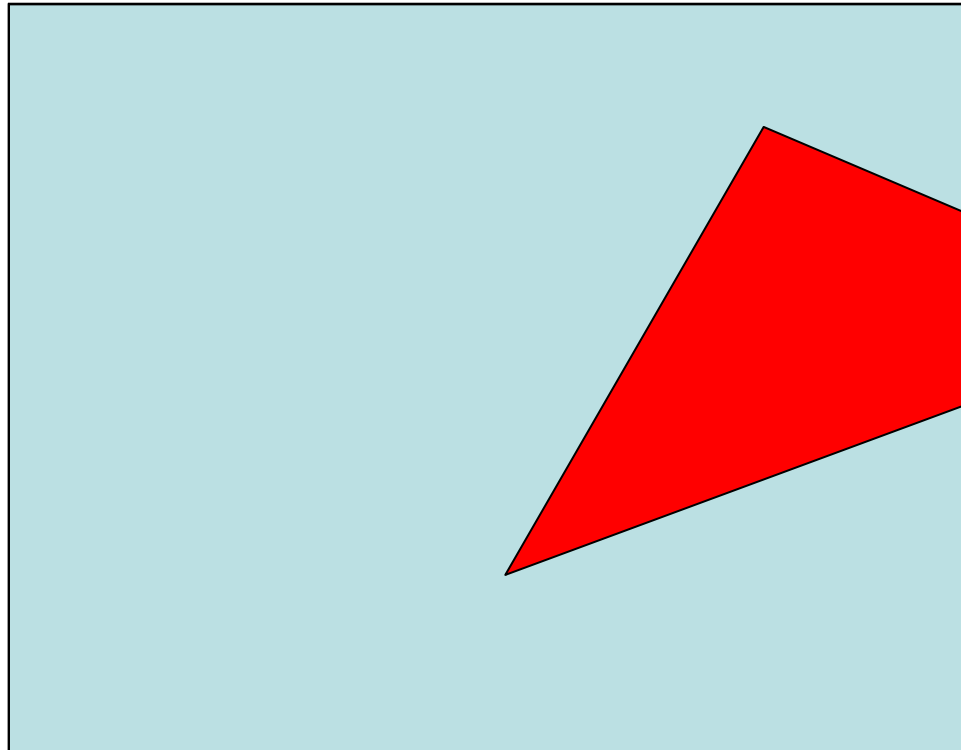
Geometry stage:

- Vertices are transformed
- Lighting can be computed
- Transformed vertices are projected to the screen
- Triangles are clipped
- Transformed to screen coordinates in preparation for rasterization

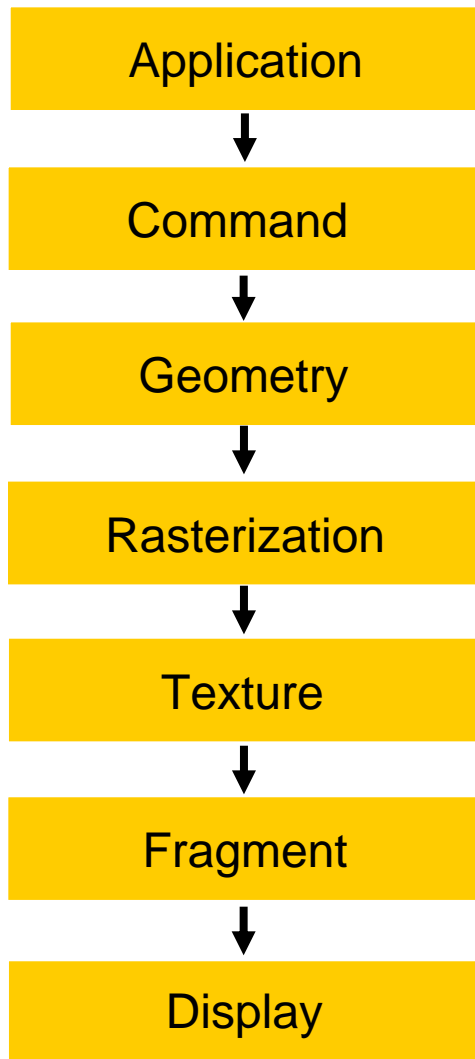
Triangle Clipping



Triangle Clipping



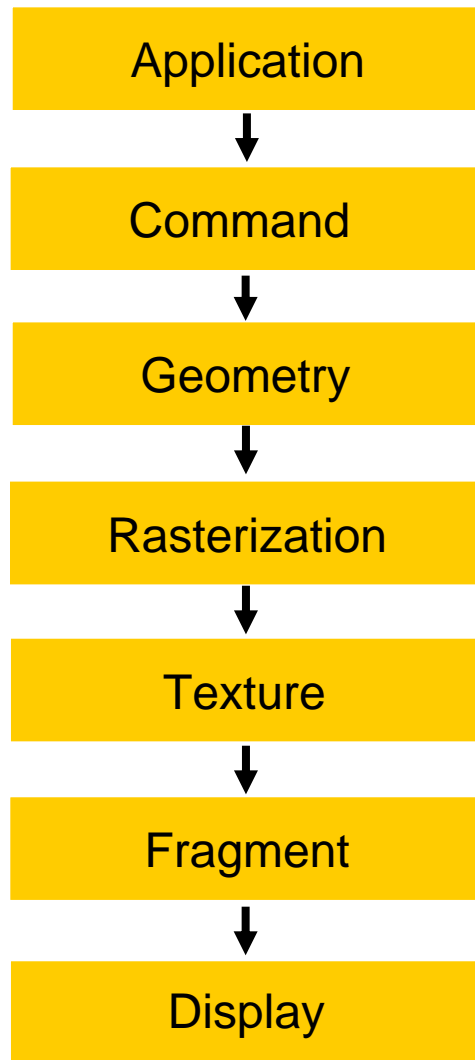
The real-time rendering pipeline



Geometry stage:

- Vertices are transformed
- Lighting can be computed
- Transformed vertices are projected to the screen
- Triangles are clipped
- Transformed to screen coordinates in preparation for rasterization

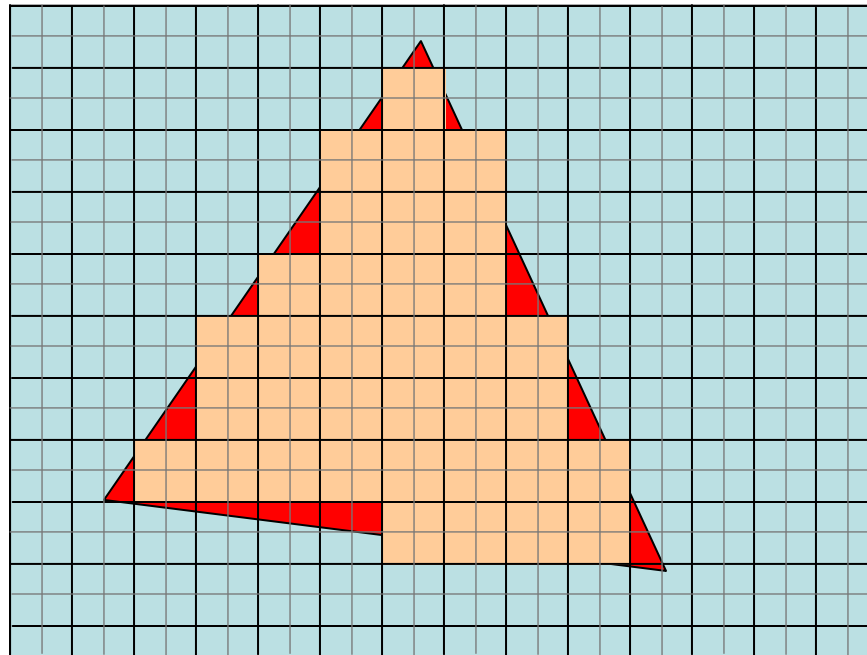
The real-time rendering pipeline



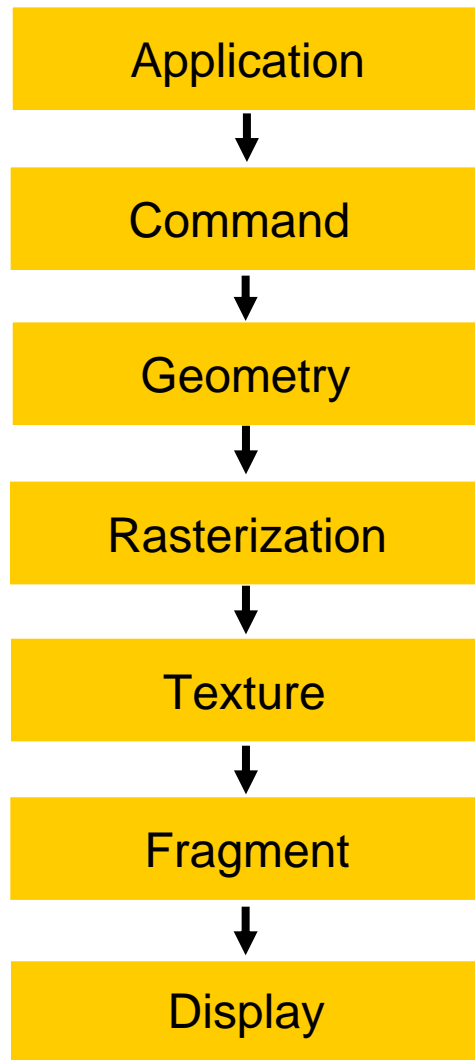
Rasterization:

- Geometrical triangles are converted into pixels
- Also called “scan conversion”

Rasterization



The real-time rendering pipeline



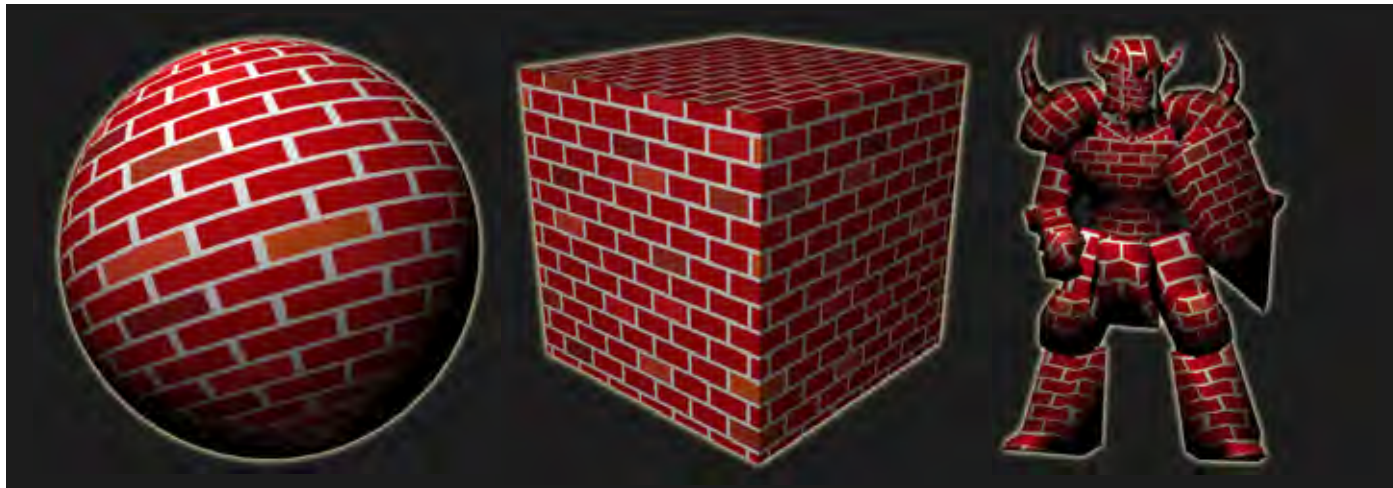
Texture:

- Allows modulation of the surface color (or other properties) by a bitmap

Texturing

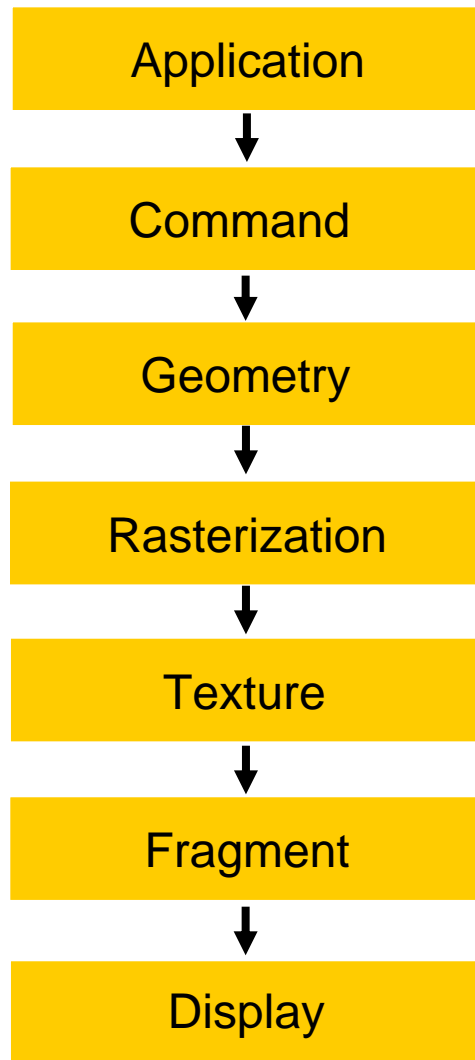


standard geometry



texture-mapped geometry

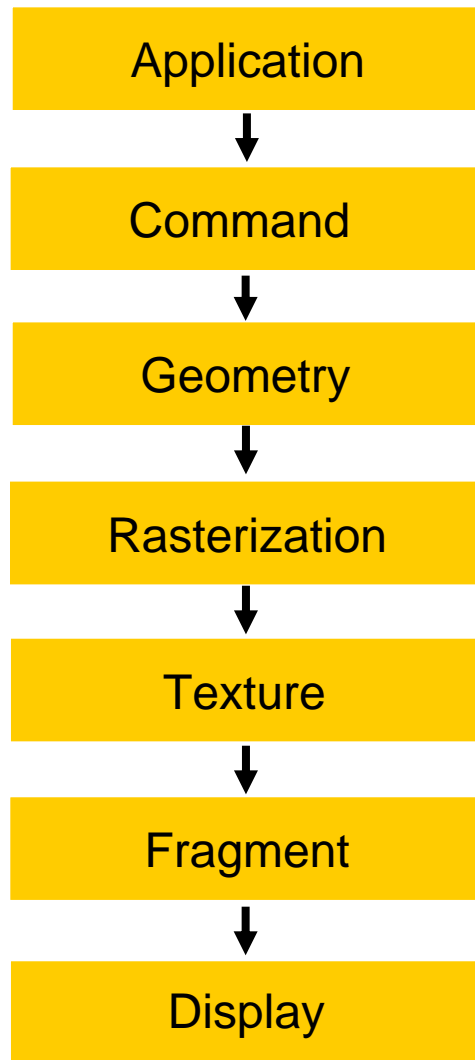
The real-time rendering pipeline



Fragment processing:

- Traditionally this was only simple blending
- Now GPUs allow for programmability
- Also includes framebuffer ops such as depth test

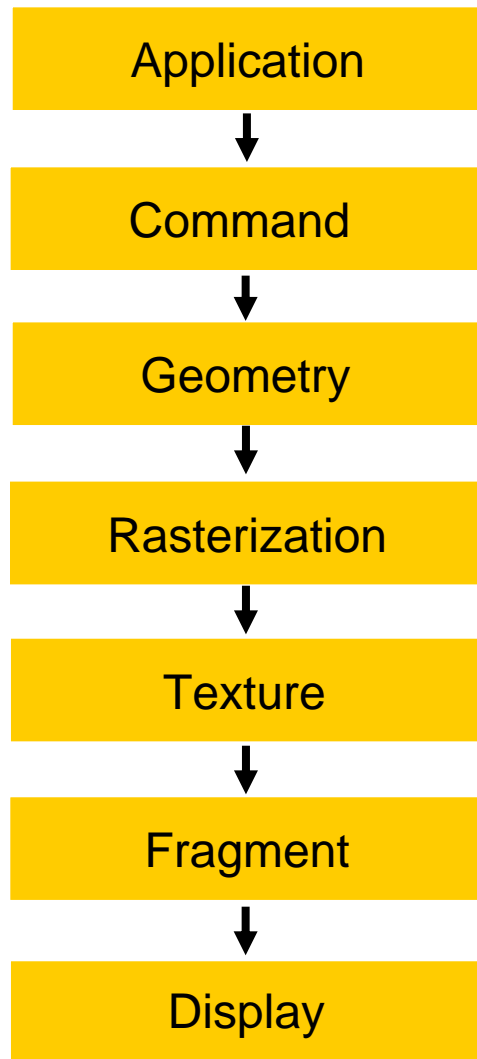
The real-time rendering pipeline



Display:

- D/A converters scan out the framebuffer to the display

The real-time rendering pipeline



OpenGL

- OpenGL is an API for 3-D graphics
- It gives us a language to “talk” to the hardware and tell it what we want it to do



OpenGL State Machine

- Programming OpenGL is like programming a state machine



Building primitives

- Use `glBegin()` to tell it what sort of primitive to build
- E.g.
 - `glBegin(GL_POLYGON);`
 - `glBegin(GL_TRIANGLES);`
 - `glBegin(GL_LINES);`

Immediate mode vertices

- `glVertex*()` – Issues a vertex to the hardware
- `glColor*()` – Issues a color that will be bound to that vertex

Information at each vertex

- Position
- Color
- Texture coordinates
- Normals
- etc.



Example

```
void drawPolygon(void) {  
    glBegin(GL_POLYGON);  
    glVertex2f(0.0, 0.0);  
    glVertex2f(0.0, 3.0);  
    glVertex2f(4.0, 2.0);  
    glVertex2f(-1.0, 1.0);  
    glEnd();  
}
```

Example

```
void drawPolygon2(void) {
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(0.0, 0.0);

    glColor3f(0.0, 1.0, 0.0);
    glVertex2f(0.0, 3.0);

    glColor3f(1.0, 0.0, 1.0);
    glVertex2f(4.0, 2.0);

    glColor3f(1.0, 1.0, 0.0);
    glVertex2f(-1.0, 1.0);
    glEnd();
}
```

Vertex arrays

- Instead of specifying the data in immediate mode (each vertex one at a time), you can specify them more quickly using vertex arrays
 - `glVertexArrayPointer()`
 - `glColorPointer()`
 - `glTexCoordPointer()`
- etc