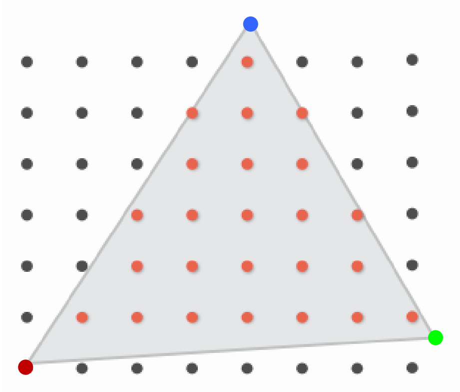
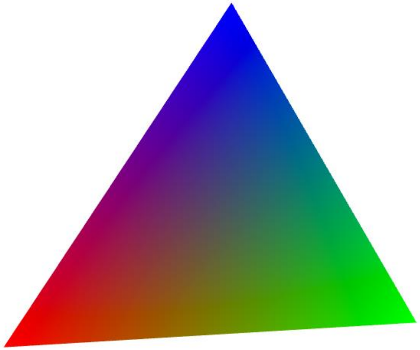
# The Graphics Pipeline

Prof. Dr. Markus Gross



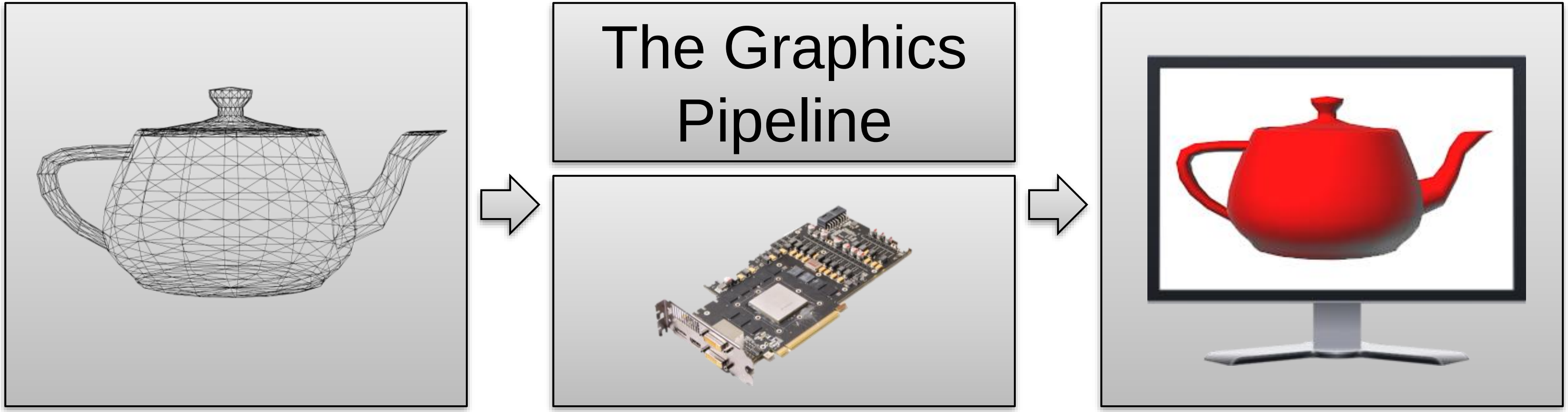


ETH zürich

cgl

# The Graphics Pipeline

- From geometry, materials, and lighting to pixels




ETH zürich

# The Graphics Pipeline

- From geometry, materials, and lighting to pixels

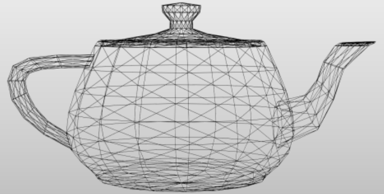
Inputs

Outputs

Geometry
Materials & Lighting
Virtual Camera

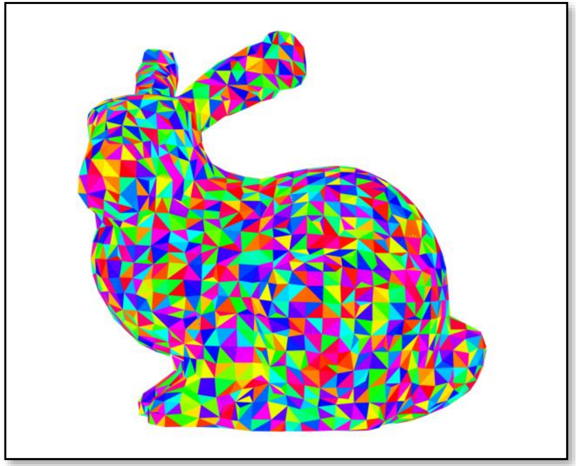
The Graphics Pipeline

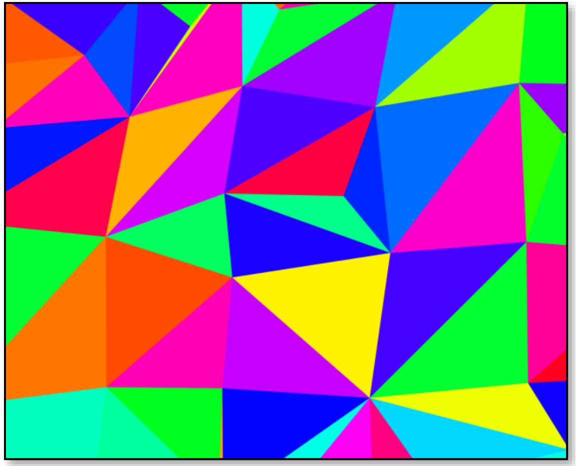
Colors for Display



# The Graphics Pipeline

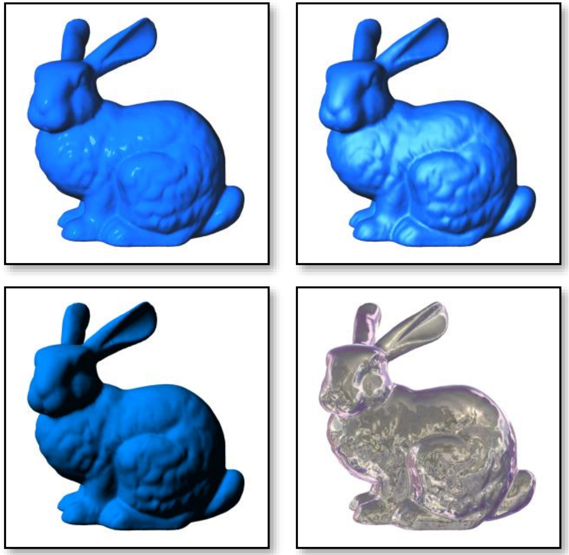
- Input: Geometry representation
- Triangles, points, lines, other primitives

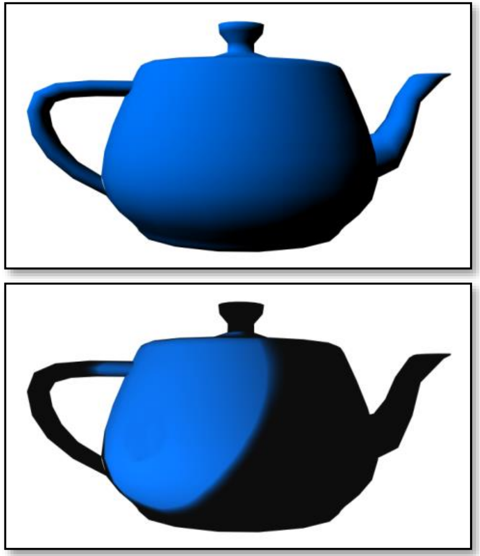




# The Graphics Pipeline

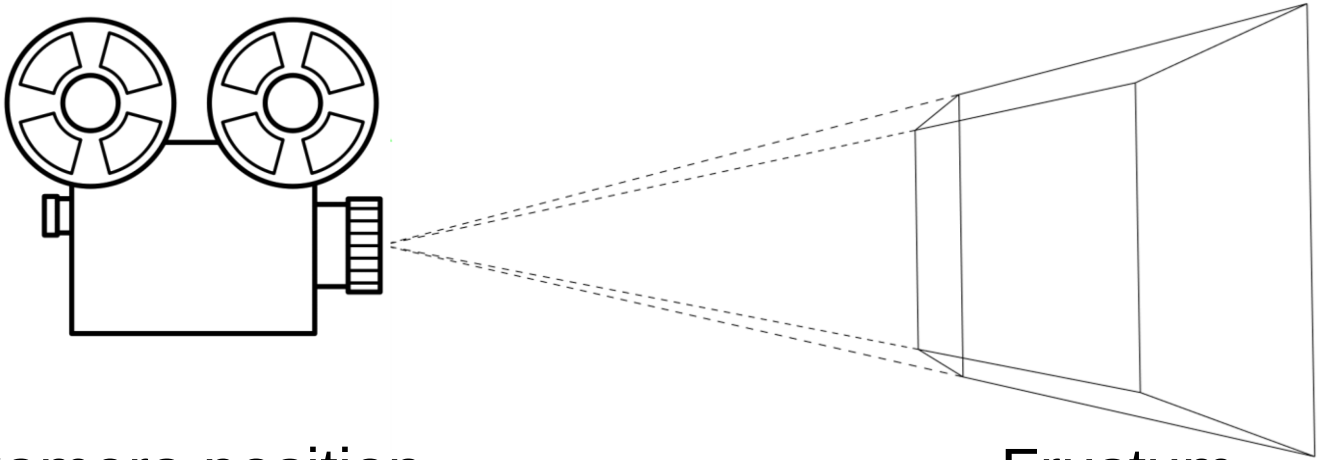
- Input: Materials & Lighting Models





# The Graphics Pipeline

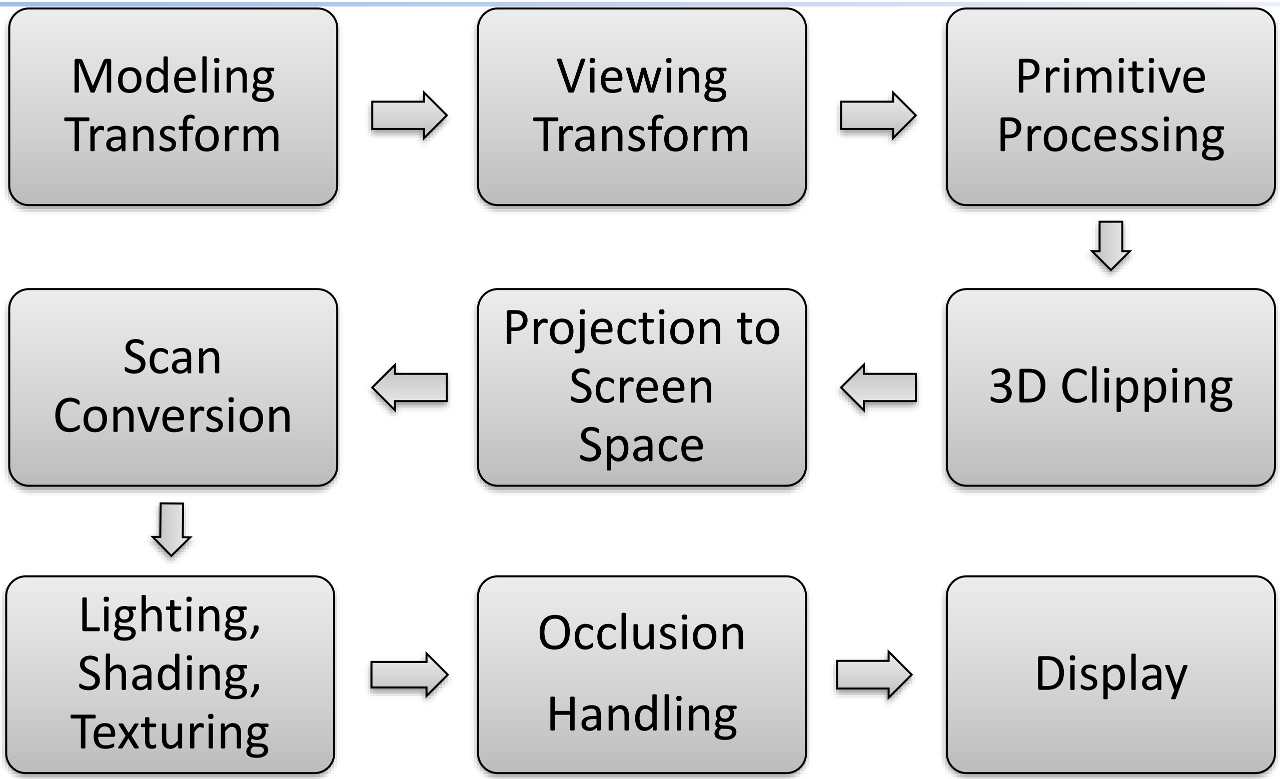
- Input: Virtual camera



Camera position

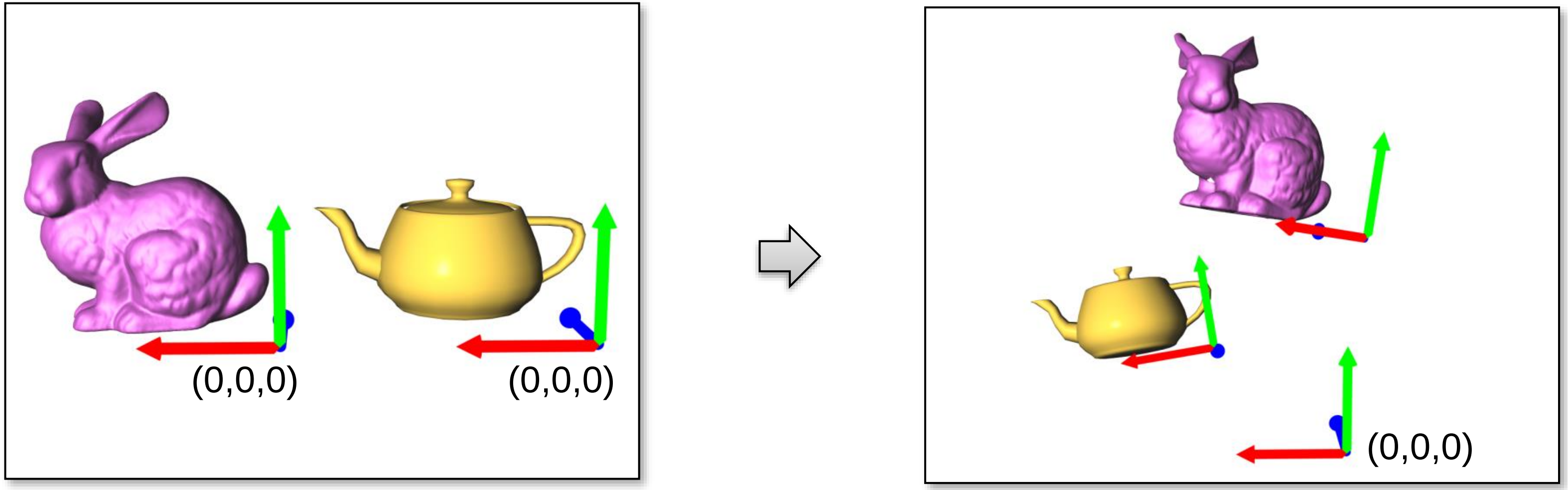
Frustum

# The Graphics Pipeline



# The Graphics Pipeline

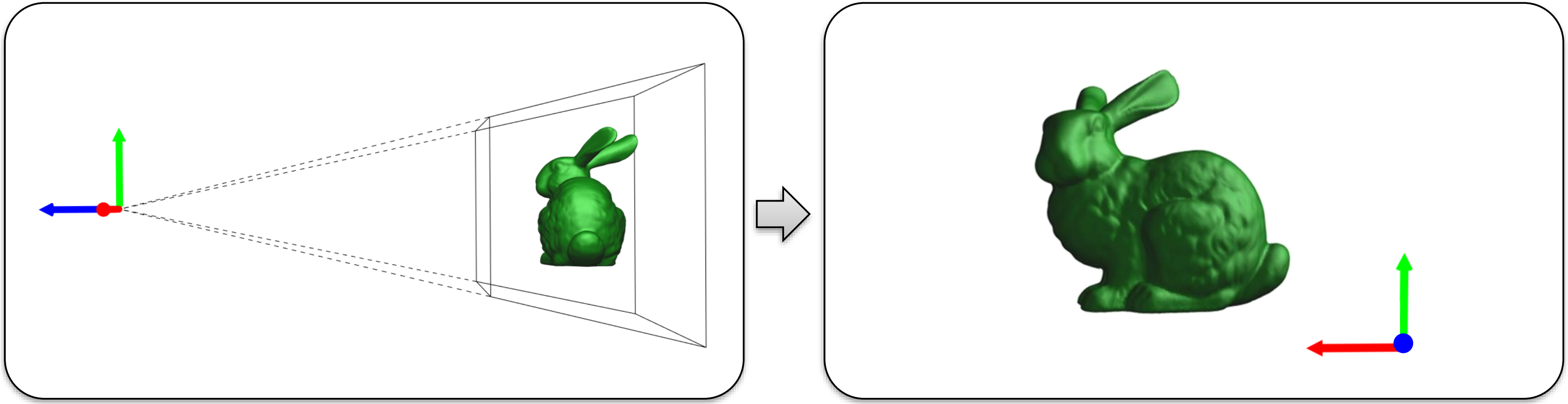
Modeling Transform




From object to world space

# The Graphics Pipeline

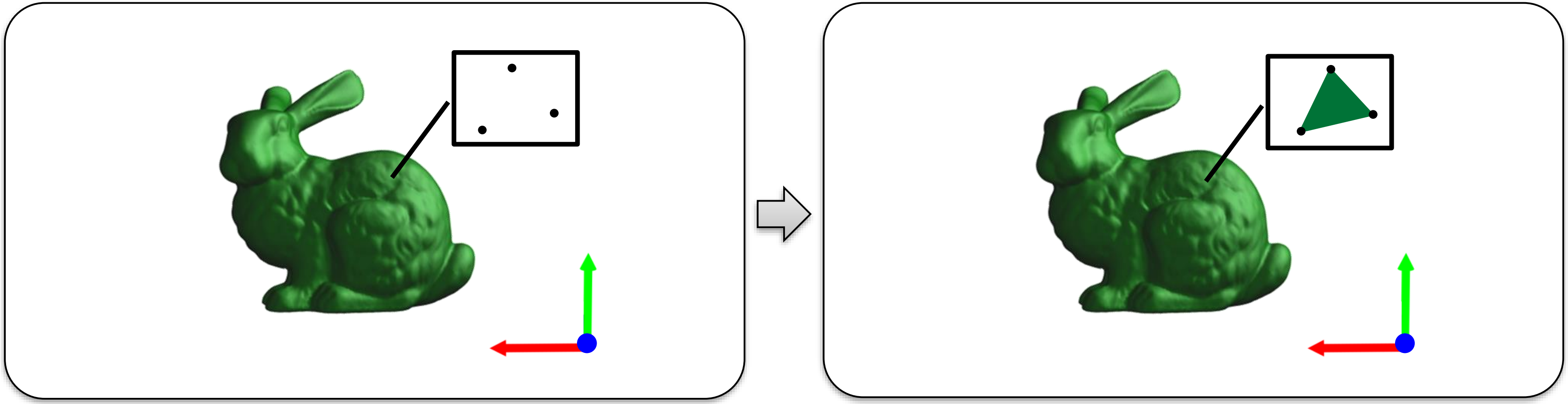
Viewing Transform



From world to camera space

# The Graphics Pipeline

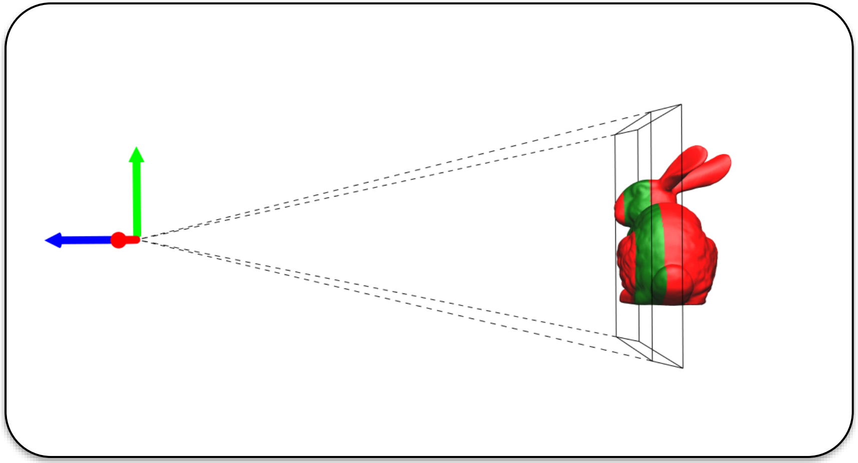
Primitive Processing

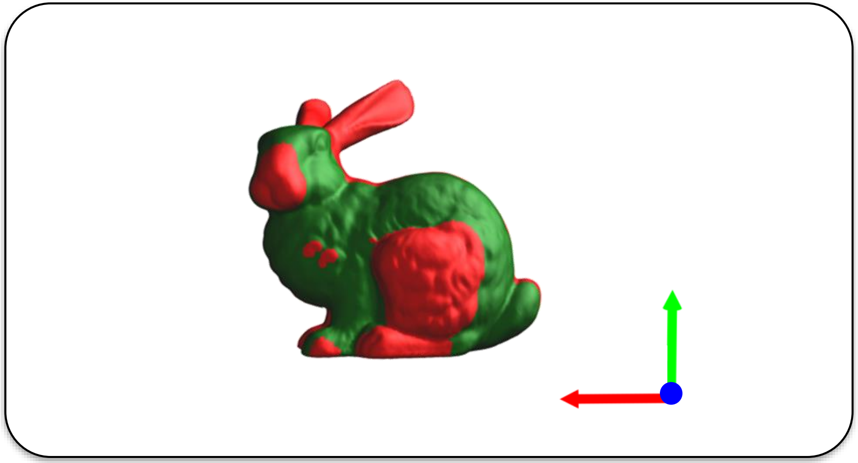


Output primitives from transformed vertices

# The Graphics Pipeline

3D Clipping





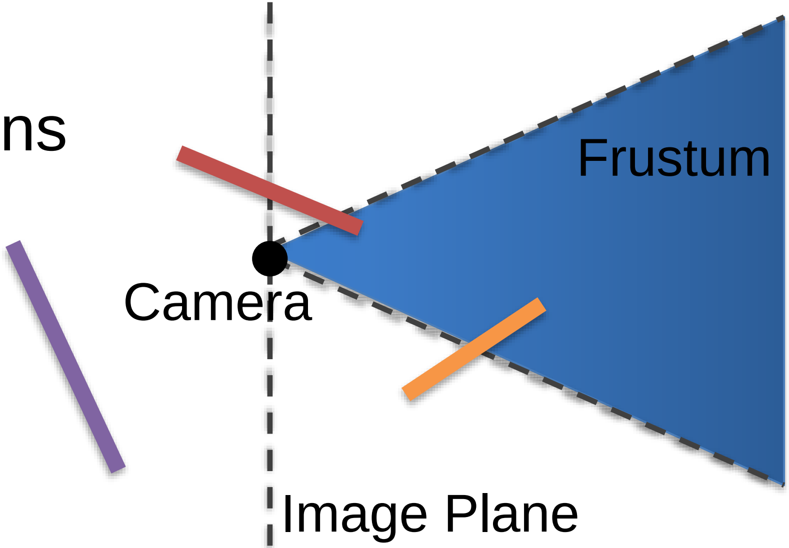
Remove parts of objects (primitives) outside the frustum

# The Graphics Pipeline

3D Clipping

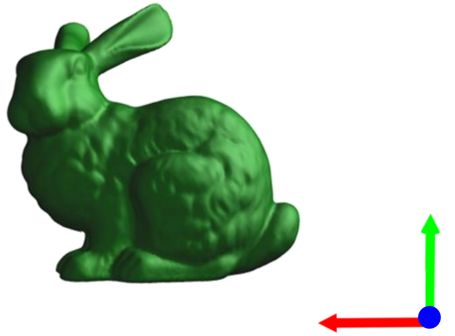
Why do we need clipping?

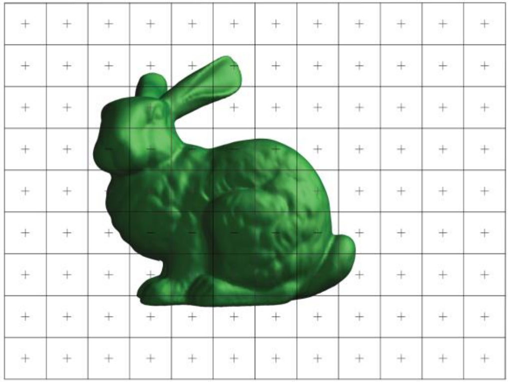
- Avoid unnecessary computations
- Avoid numerical instabilities



# The Graphics Pipeline

Projection to Screen Space





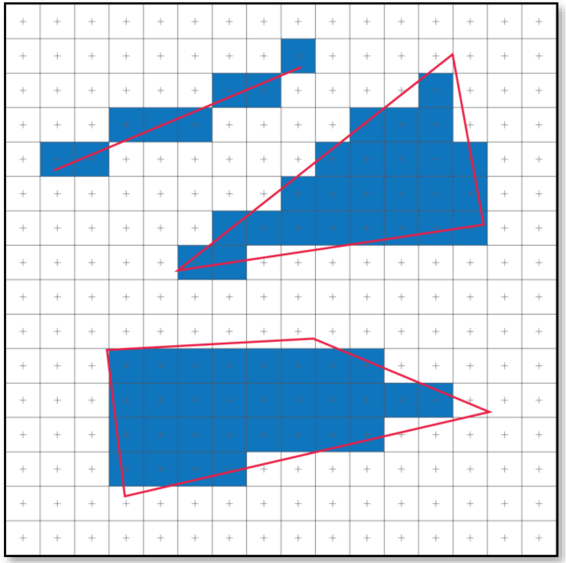
Screen space

Project from 3D to 2D screen space

# The Graphics Pipeline

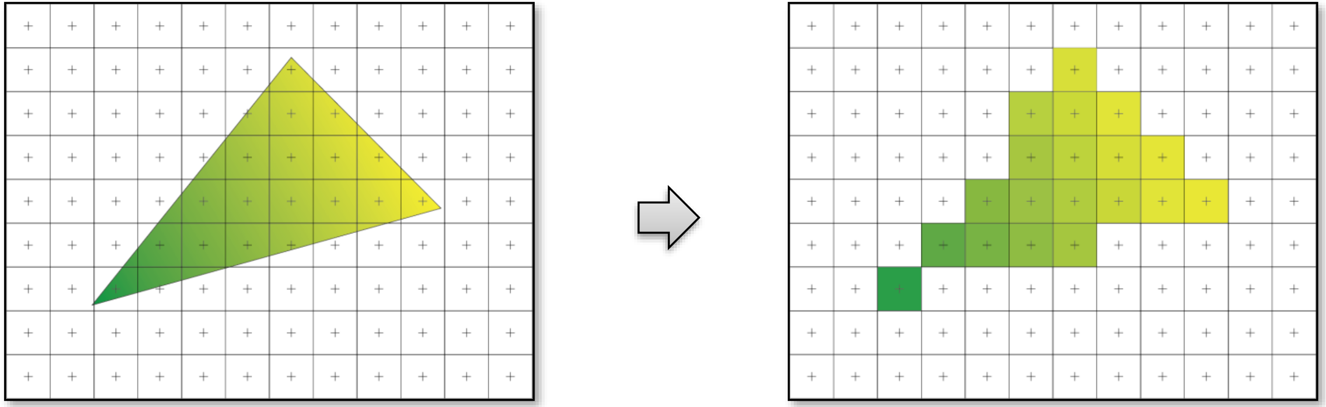
Scan Conversion

- Discretize continuous primitives
- Triangles, lines, polygons



# The Graphics Pipeline

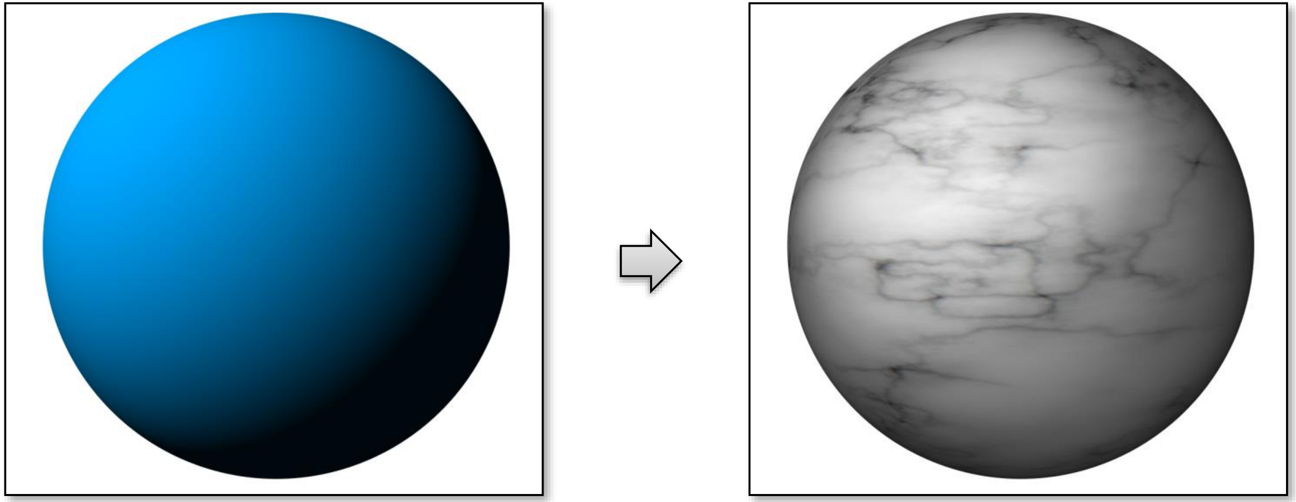
Scan Conversion



- Interpolate attributes at all covered samples (normal, depth, uv)

# The Graphics Pipeline

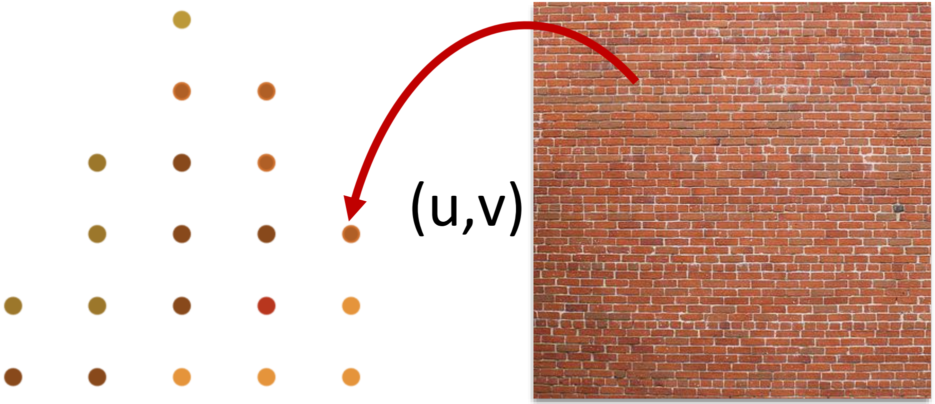
Lighting, Shading, Texturing



- Compute color based on lighting, shading, texture map

# The Graphics Pipeline

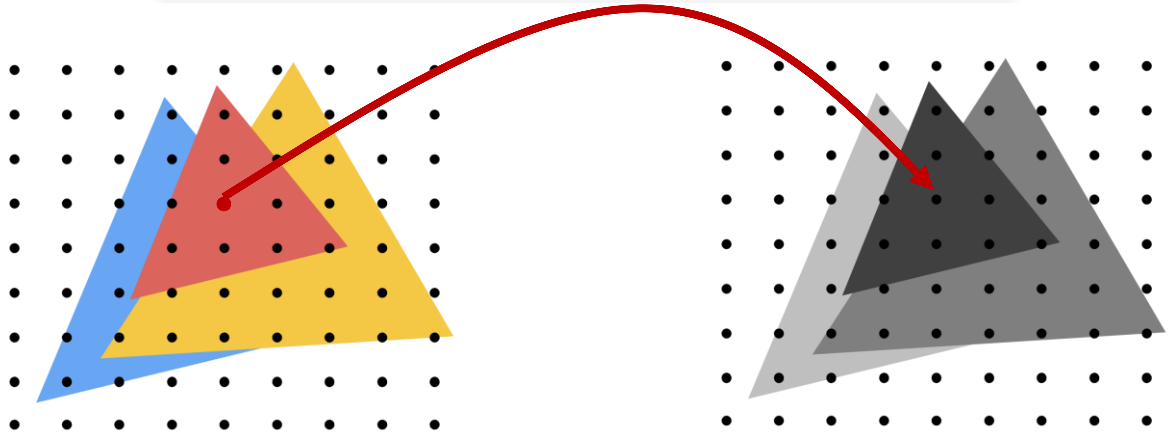
Lighting, Shading, Texturing




- Texture mapping using uv coordinate

# The Graphics Pipeline

Occlusion Handling



Color buffer

Z-buffer

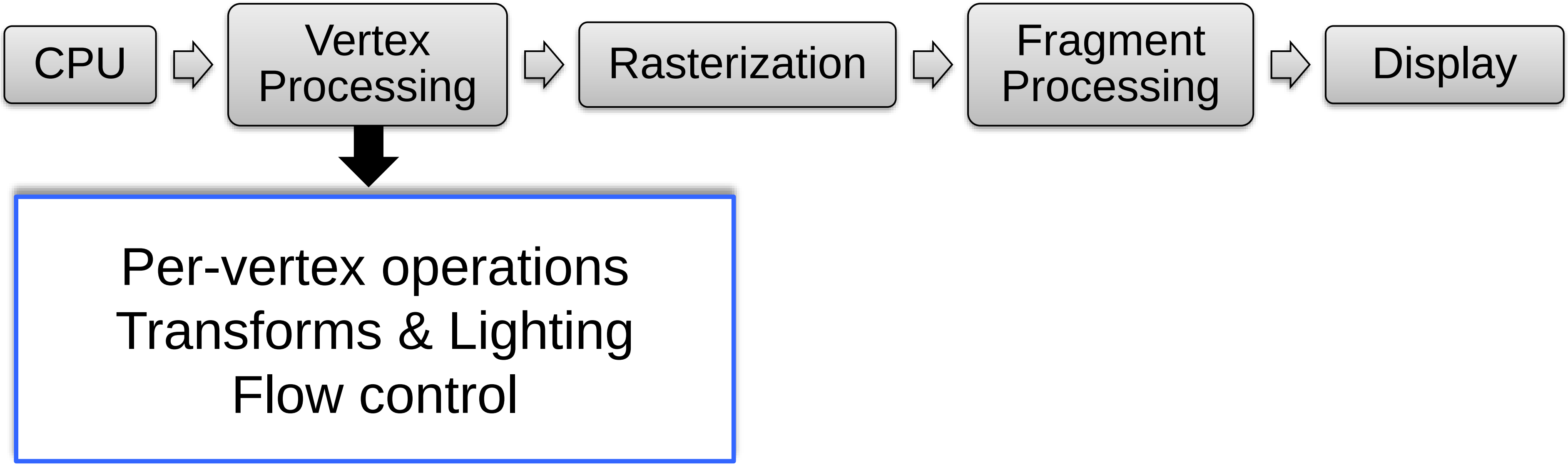
- Update color buffer using the depth buffer (Z-buffer)

# Programmer’s View

- Contemporary pipeline

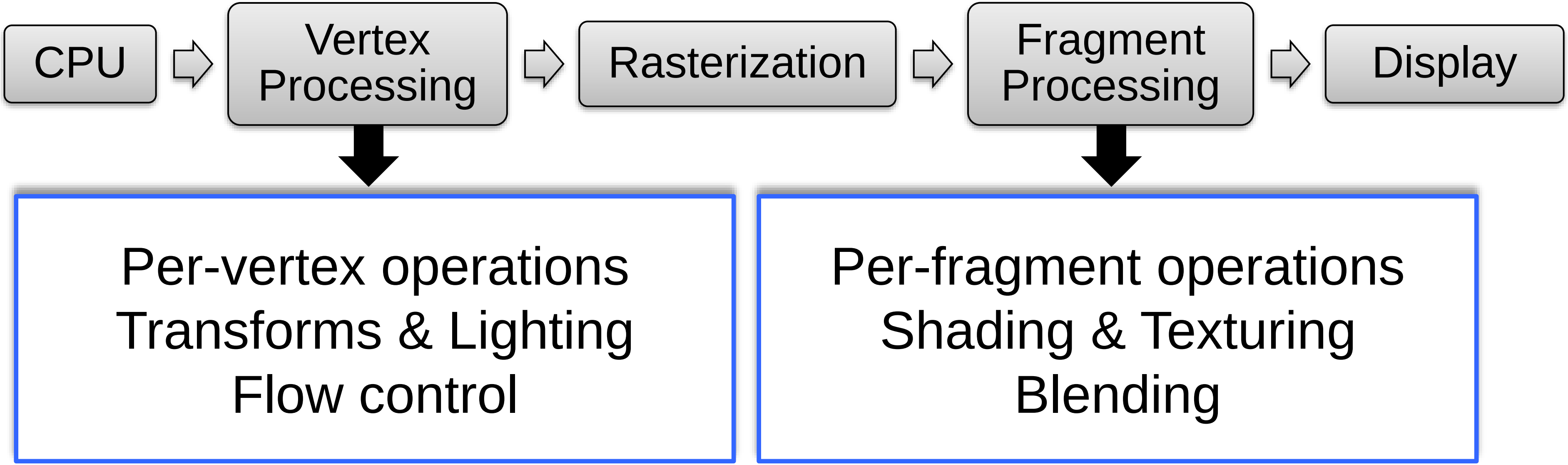
# Programmer's View

- Contemporary pipeline



# Programmer's View

- Contemporary pipeline



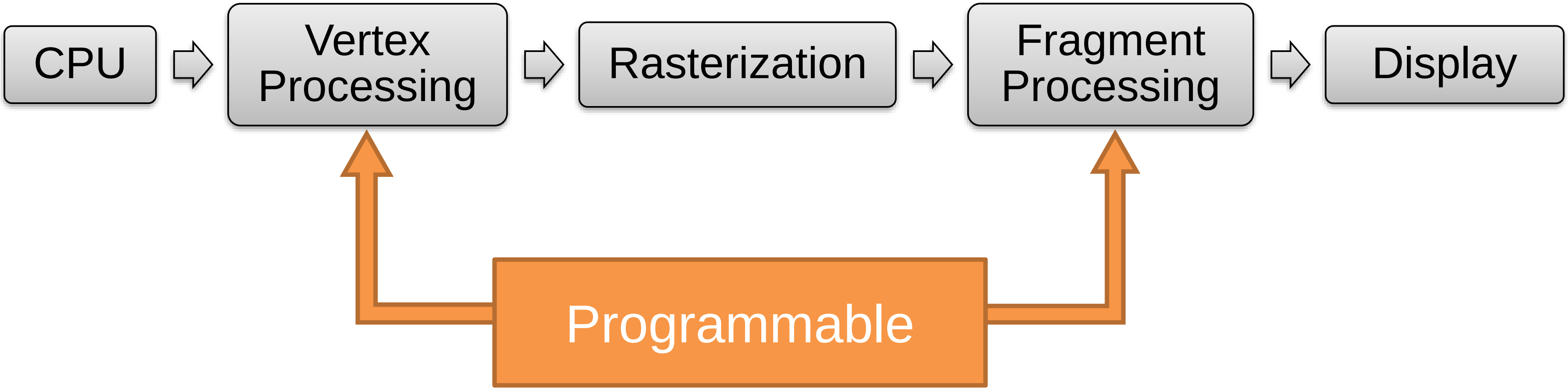
# Programmer's View

- Contemporary pipeline


- Historically: Hardwired floating point operations, fixed point
- Now: Programmable, complex floating point operations
- 1 billion vertices / sec. & 50 billion fragments / sec.

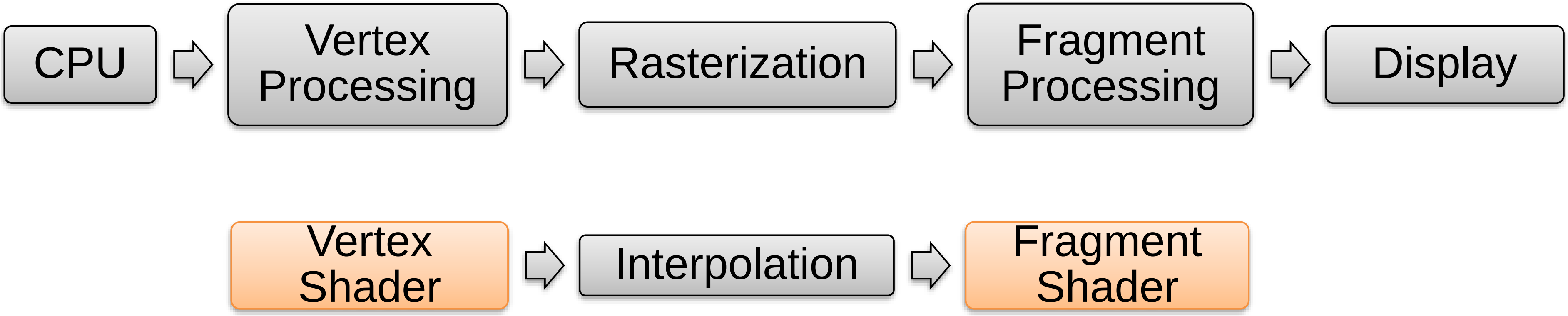
# Programmer’s View

- Contemporary pipeline



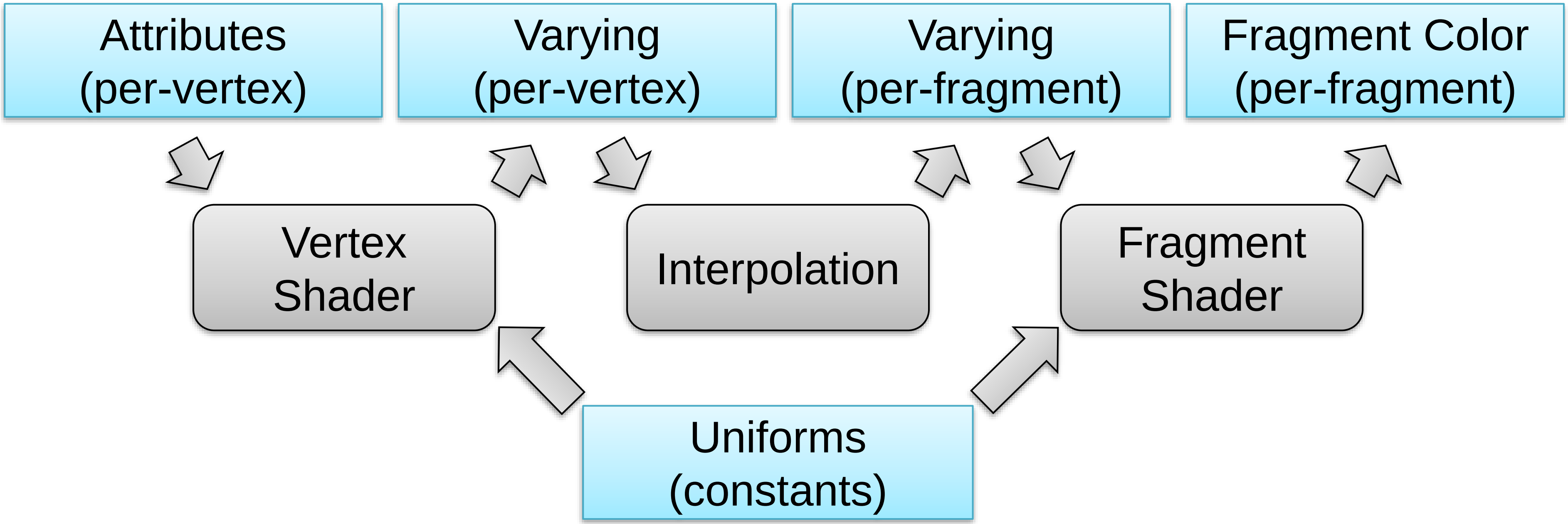
# Programmer’s View

- Programming with “shaders”

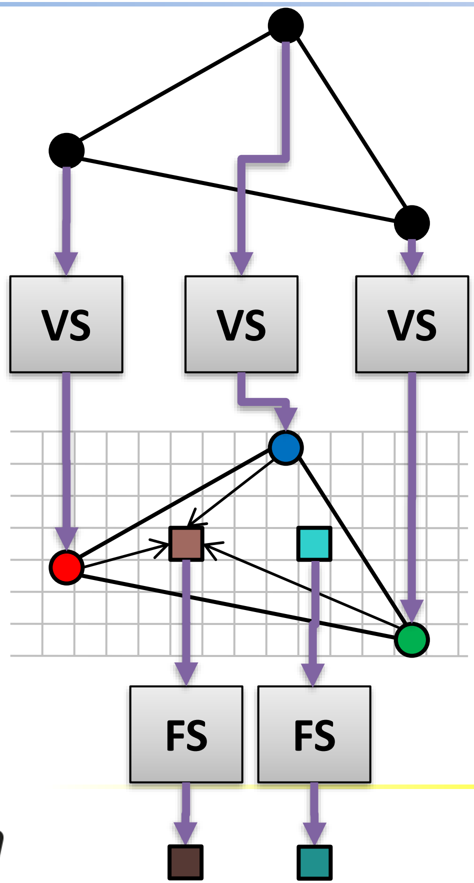


# Programming with Shaders

- Programming with “shaders”



# Programming with Shaders




**Attributes** given per vertex

Vertex Shader computes **varying**

Interpolate **varying** values

Fragment Shader computes pixel color

# Programmer's View

- Vertex Shader

```
uniform mat4 modelViewMatrix;
uniform vec3 lightPosition;
attribute vec3 pos;
attribute vec3 vertexNormal;

varying vec3 lightDirection, normal;

void main()
{
        vec4 vertexPosition = modelViewMatrix * pos;
        lightDirection = vec3(lightPosition - vertexPosition);
        normal = vertexNormals;
        gl_Position = vertexPosition;
}
```

# Programmer’s View

- Vertex Shader

```
uniform mat4 modelViewMatrix;
uniform vec3 lightPosition;
attribute vec3 pos;
attribute vec3 vertexNormal;

varying vec3 lightDirection, normal;

void main()
{
        vec4 vertexPosition = modelViewMatrix * pos;
        lightDirection = vec3(lightPosition - vertexPosition);
        normal = vertexNormals;
        gl_Position = vertexPosition;
}
```

Variables to be interpolated and passed to fragment shader

# Programmer's View

- Vertex Shader

```
uniform mat4 modelViewMatrix;
uniform vec3 lightPosition;
attribute vec3 pos;
attribute vec3 vertexNormal;

varying vec3 lightDirection, normal;

void main()
{
        vec4 vertexPosition = modelViewMatrix * pos;
        lightDirection = vec3(lightPosition - vertexPosition);
        normal = vertexNormals;
        gl_Position = vertexPosition;
}
```

Computing per-vertex attributes

# Programmer's View

- Fragment Shader

```
varying vec3 lightDirection, normal;

void main()
{
		vec3 lightDirectionNormalized = normalize(lightDirection);
		float intensity = dot(lightDirectionNormalized, normal);
		gl_FragColor = vec4(intensity, intensity, intensity, 1.0);
}
```

# Programmer's View

- Fragment Shader

```
varying vec3 lightDirection, normal;

void main()
{
		vec3 lightDirectionNormalized = normalize(lightDirection);
		float intensity = dot(lightDirectionNormalized, normal);
		gl_FragColor = vec4(intensity, intensity, intensity, 1.0);
}
```

Interpolated variables passed from vertex shader

# Programmer's View

- Fragment Shader

```
varying vec3 lightDirection, normal;

void main()
{
        vec3 lightDirectionNormalized = normalize(lightDirection);
        float intensity = dot(lightDirectionNormalized, normal);
        gl_FragColor = vec4(intensity, intensity, intensity, 1.0);
}
```

Computing the color of this fragment (gl_FragColor)

# Programmer’s View

- Fragment ≠ Pixel
- A fragment can store
  - Color
  - Position
  - Depth
  - Texture Coordinates
  - Window ID
  - …

# Graphics APIs

- Application Programming Interfaces

# Graphics APIs

- Application Programming Interfaces
- Access to the graphics hardware
- Hardware-independent
- Abstract away complex details

# Graphics APIs

- Application Programming Interfaces

PIXAR's RenderMan

Open Inventor®

- Describe the scene
- Scene graph

OpenGL™

Microsoft® DirectX® 11

- Sequence of drawing commands
- More direct control

# OpenGL

- **O**pen **G**raphics **L**ibrary
- Initially defined by Silicons Graphics Inc.
- Since 2006: managed by Khronos Group
- OpenGL Architectural Review Board (ARB)

…more

# OpenGL

- **O**pen **G**raphics **L**ibrary
- Platform-independent
- Available on many platforms

Microsoft Windows  Mac OS X  Linux  solaris  …more

# OpenGL™

- Updates automatic by GPU drivers
- Language bindings
  - C++  Java  Ada  Fortran  Perl  Python
- Versions and variants

OpenGL|ES™ OpenVG™ WebGL OpenCL •••