COMPSCI 715 Part 2

Lecture 1 - Introduction to Shaders

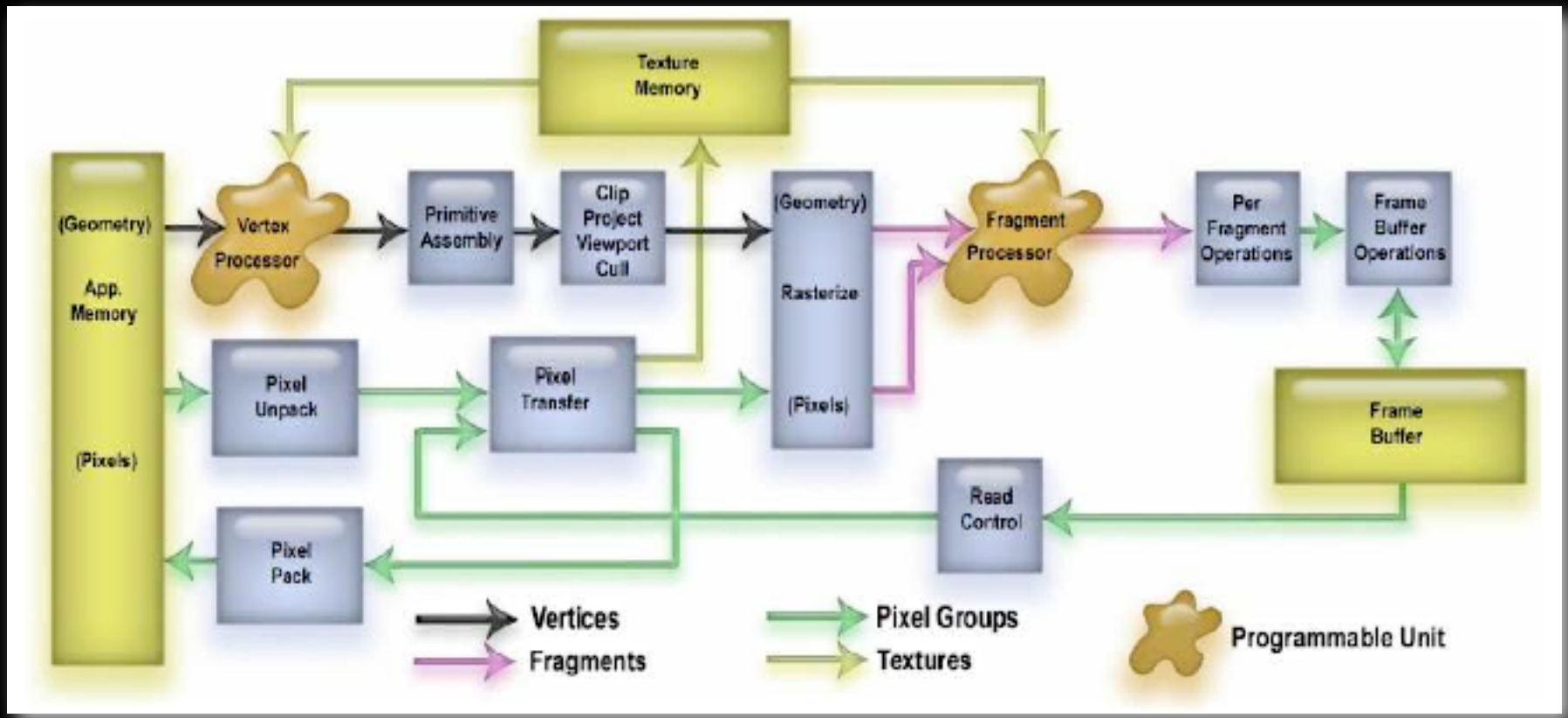
Topics in Part 2

- GLSL (OpenGL Shading Language)
- Physically based modelling / rendering
- Advanced Rendering Techniques
- SIGGRAPH Project Presentations

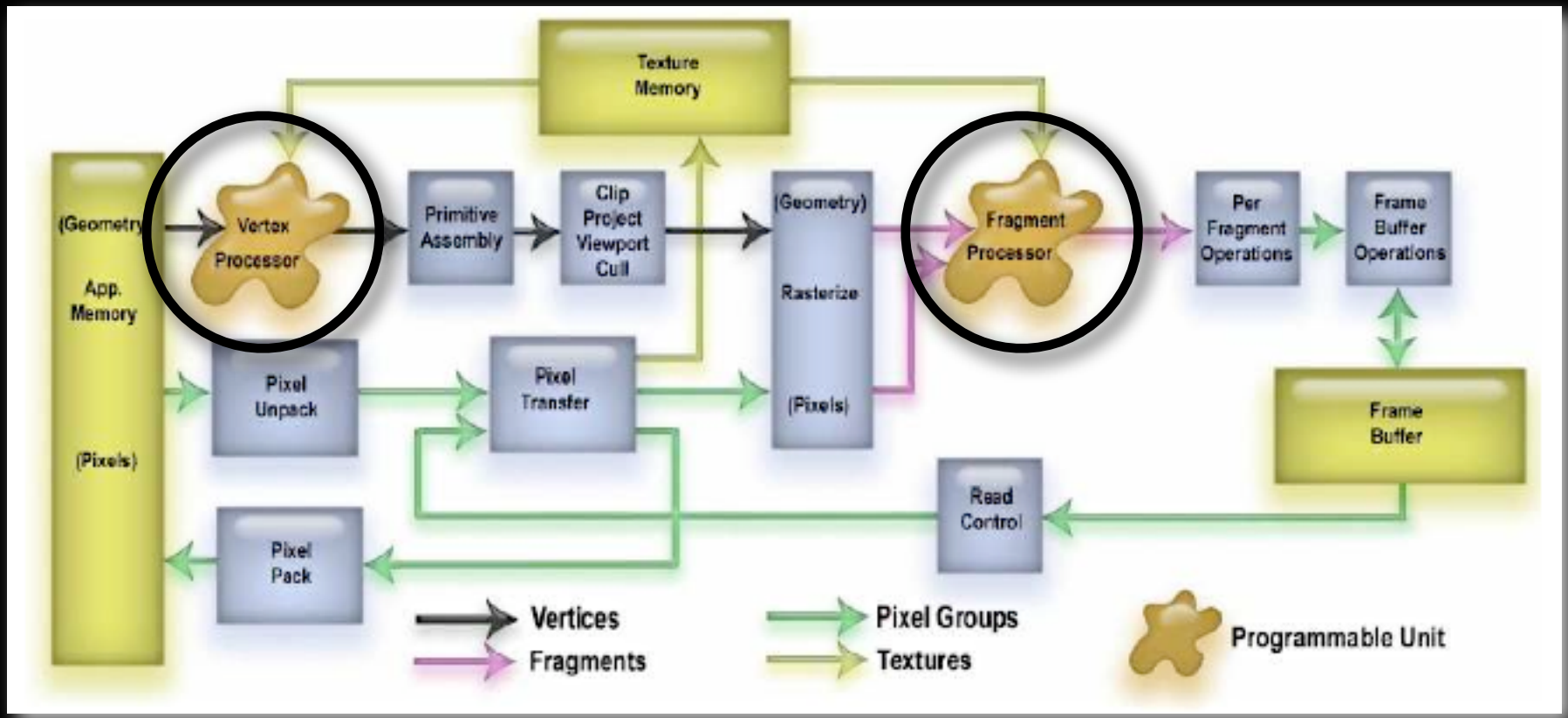
History of Shaders

- Originally fixed functionality
- Registry combinations in rasterizer
- Assembly code injection
- High level langauges
  - All designed as C-style languages

Hardware Pipelines



Programmable Pipelines



CPU vs GPU

- Highly optimized vector calculations

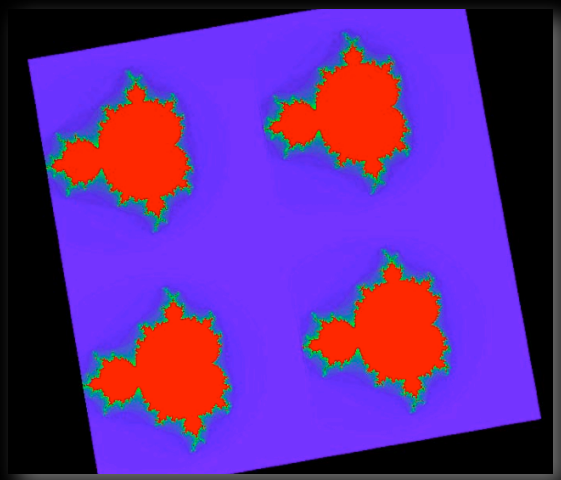
  vs.

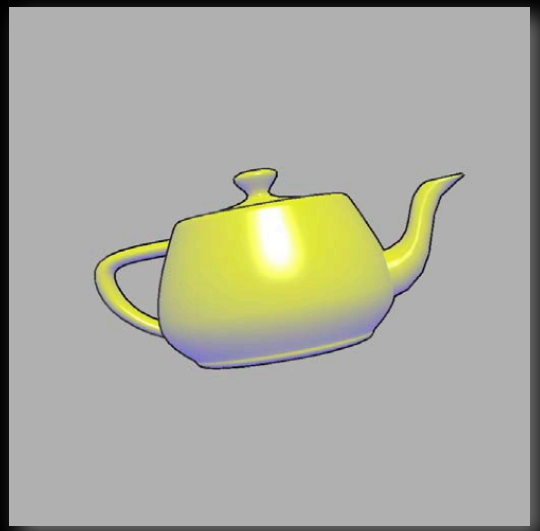
- Flexibility

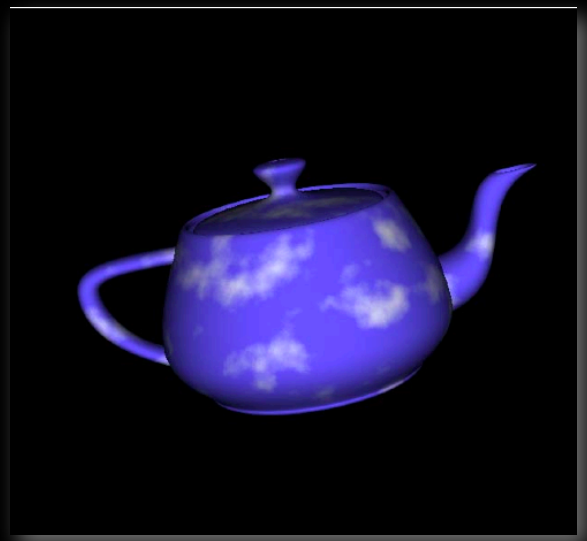
Things you can do

- Infinite LOD (think fractals)
- Realistic materials (wood, stone etc)
- Natural Phenomena (fire, clouds etc)
- Non-photorealistic effects
- Surface effects / lighting

Things you can do







DEMO

Languages

- Assembly
- High Level Languages
  - HLSL
  - CG
  - GLSL

Assembly

- Graphics Card/Vendor specific
- Similar analogy to assembly languages such as Alpha or X86

HLSL

- Microsoft Direct X
- Easy to check / document card compliance
- Limited to Windows OSes

CG

- Vendor Specific / NVIDIA but developed in cooperation with Microsoft
- Syntactically very similar to HLSL
- Converted to DirectX / OpenGL shader assembly then passed through API
  - Therefore NOT limited to NVIDIA hardware

GLSL

- OpenGL / Architecture Review Board
- NOT designed to be compatible with old hardware
- Driver must compile/link/optimize

What they share

- All C-Style languages including methods
- All limit access to texture memory / hardware
- All allow programmability in the vertex and fragment portions
- All are limited by the rest of the fixed pipeline

GLSL Vertex Shader

- Operate on every vertex
- Transformations not applied
- Normals not rescaled or normalized
- No automatic per-vertex lighting
- Clipping, front face, viewport mapping done on output

GLSL Fragment Shader

- Fragment == Potential Pixel
- Instead of normal colour/te
- xture/blending operations
- Clamping/format conversion done after
- Highly parallel

Simple Vertex Shader

```
varying vec3 Normal;
void main( void)
{
   gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;
   Normal = normalize(gl_NormalMatrix*gl_Normal);
}
```

Simple Fragment Shader

```
varying vec3 Normal;
uniform vec3 LightColor;
uniform vec3 LightPos;

void main(void)
{
  vec3 color = LightColor;
  float dif = dot(normalize(Normal),LightPos));
  color *= max(0.0,dif);
  gl_FragColor = vec4(color,1.0);
}
```

GLSL Types

- float, int, bool
- vectors (vec3)
- matrices (mat3), float only

- sampler(n)D, sampler(n)DShadow

GLSL Type Qualifiers

- const
- attribute - per vertex data values
- uniform - set from within the program
- varying - interpolated between vertices

GLSL Specials

- Vertex:
  - vec4 gl_Position //MUST write this
  - vec4 glClipPosition
  - float gl_PointSize
  - vec4 gl_FragColor //To set the color

GLSL Built in Attributes

- gl_Vertex
- gl_Normal
- gl_Color
- gl_SecondaryColor
- gl_MultiTexCoord(n)
- gl_FogCoord

GLSL Built in Uniforms

- gl_ModelViewMatrix
- gl_ProjectionMatrix
- gl_DepthRange
- gl_ClipPanes

GLSL Functions

- Trigonometry (sin, cos, asin, acos)
- Exponentials (pow, exp2, log2, sqrt)
- Common (abs, sign, floor, fract, clamp)
- Interpolations (mix, step, smoothstep)
- Geometric (length, dot, cross, reflect)

GLSL Functions

- Texture(texture1D, texture2D)
- Shadow(shadow2D, shadow2D)
- Special (ftransform, dFdx, dFdy)
- Noise

GLSL reserved types

- half hvec2, hvec3, hvec4
- fixed fvec2, fvec3, fvec4
- double dvec, dvec3, dec4
- sampler2DRect

Driver Compatibility

- Very few cards support full shader spec
- Some functions may be handed off to software
- As said before: not designed for backwards compatibility

Lighting Vertex Shader

```
varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;
void main( void)
{
    vec4 ecPos = gl_ModelViewMatrix*gl_Vertex;
    lightVec = vec3(normalize(gl_LightSource[0].position
                    - ecPos));
    viewVec = vec3(normalize(-ecPos));
     normal = normalize(gl_NormalMatrix*gl_Normal);
     gl_Position = ftransform();
}
```

Lighting Fragment Shader

```
varying vec3 normal;
varying vec3 lightVec;
varying vec3 viewVec;

void main(void)
{
  vec3 nnormal = normalize(normal);
  vec3 nlighVec = normalize(lightVec);
  vec3 nViewVec = normalize(viewVec);
  vec3 reflectVec = reflect(-nLightVec,nnormal);


}
```

Lighting Fragment Shader

```
float specVal = clamp(dot(reflectVec,nViewVec),0.0,1.0);
specVal = pow(specVal,6.0);

gl_FragColor = vec4(
(gl_LightSource[0].diffuse*gl_Color*max(dot(nlightVec,nnormal,0.0)
+
(gl_LightSource[0].specular*gl_Color*specVal)).rgb
,1.0);
```

Sources

- Rost, Randi (2005) Introduction to the OpenGL Shading Language, 3DLabs Presentation
- Lovesey, Anthony (2005) A Comparison of Real Time Graphical Shading Languages, Technical Report University of New Brunswick