MIC-GPU: High-Performance Computing for Medical Imaging on Programmable Graphics Hardware (GPUs)

Cg Programming

Klaus Mueller, Wei Xu, Ziyi Zheng — Computer Science, Center for Visual Computing, Stony Brook University, NY

Fang Xu — Siemens USA Research, Princeton, NJ

Content

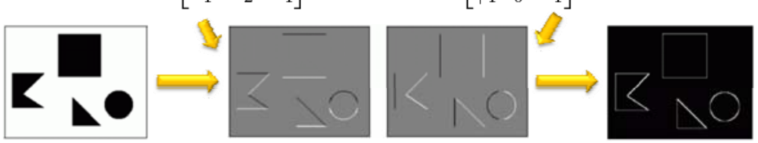
- Example Program
- Graphics Hardware Pipeline
- Introduction to Cg
- Cg Programming Examples
  - Multitextures vs. Dependent Textures
  - Render-To-Texture vs. Framebuffer Object (FBO)
  - Multi-pass vs. Single-pass
  - Multiple Render Target (MRT)
  - Early Fragment Kill (Early Z-test)

Example program

Edge Detection using Sobel operator

- A discrete differentiation operator
- The approximation of the gradient of the intensity image
- A pair of 3x3 convolution masks: gradient in the x-direction and gradient in the y-direction

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$



- The approximate magnitude of the gradient is then calculated:

$$|G| = |Gx| + |Gy|$$

Example program

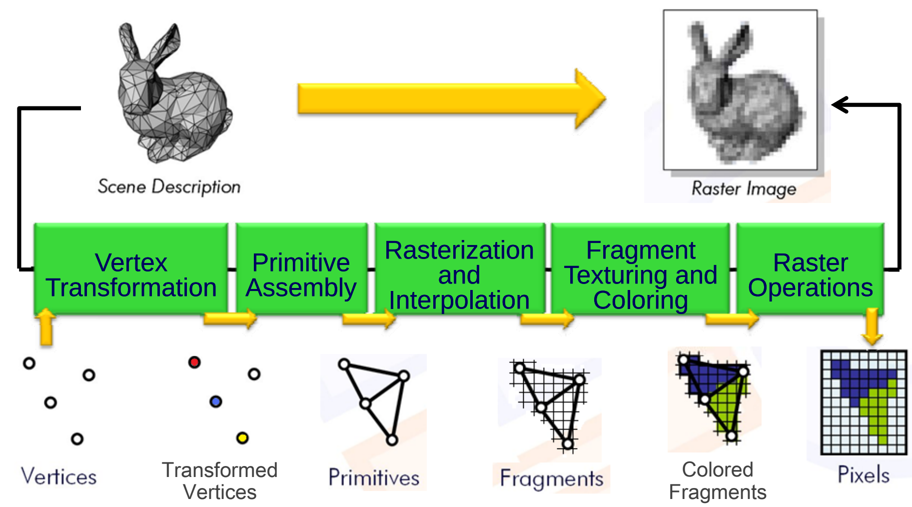
CPU code:

GPU code:

Content

- Example Program
- Graphics Hardware Pipeline
- Introduction to Cg
- Cg Programming Examples
  - Multitextures vs. Dependent Textures
  - Render-To-Texture vs. Framebuffer Object (FBO)
  - Multi-pass vs. Single-pass
  - Multiple Render Target (MRT)
  - Early Fragment Kill (Early Z-test)

Graphics Pipeline: Overview



Stage 1: Vertex Transformation

Vertex Transformation → Primitive Assembly → Rasterization and Interpolation → Fragment Texturing and Coloring → Raster Operations

Vertices | Transformed Vertices | Primitives | Fragments | Colored Fragments | Pixels

Vertex attributes:

- Position, color, texture coordinate(s), normal vector, ...

Operations:

Perform a sequence of math operations on each vertex

- Transform current position into screen position
- Generate texture coordinates for texturing
- Perform vertex lighting to determine its color

Stage 2: Primitive Assembly

Vertex Transformation → Primitive Assembly → Rasterization and Interpolation → Fragment Texturing and Coloring → Raster Operations

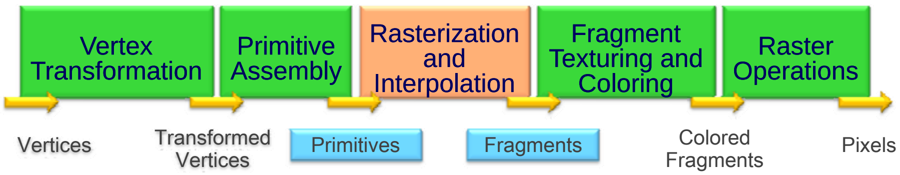
Vertices | Transformed Vertices | Primitives | Fragments | Colored Fragments | Pixels

Operations:

- Assemble vertices into geometric primitives (triangles, lines, points)

- Clipping: clip to the view frustum (the view's visible region) and other clip planes
- Culling: eliminate backward-facing polygons

Stage 3: Rasterization and Interpolation



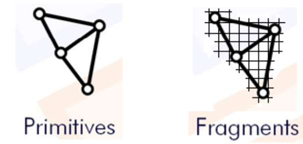
What is a fragment?

- The same size as pixel, but not yet a pixel
- Pixel: the contents of the frame buffer at a specific location
- Fragment: the state required potentially to update a particular pixel
- *Potential* pixel (whether passes rasterization tests)
- Values: color, depth, location, texture coordinate sets, ...

Stage 3: Rasterization and Interpolation

Rasterization:

- Rasterizes geometric primitives into fragments
- Determines the set of fragments covered by a primitive



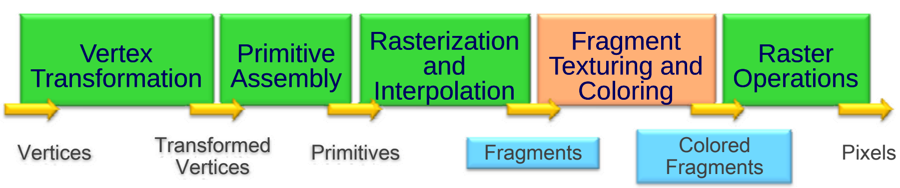
- Yields a set of pixel locations and fragments

Number of vertices and fragments are unrelated

Interpolation

- Determines fragment values from vertices

Stage 4: Fragment Processing



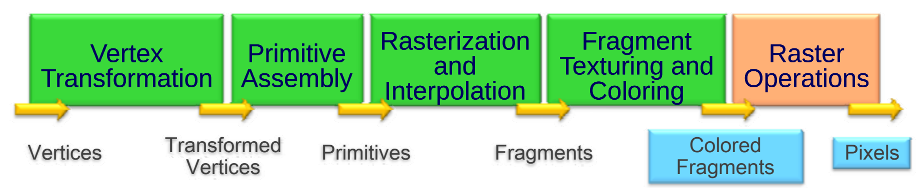
Operations:

- Texturing and coloring
- Math operations

Output:

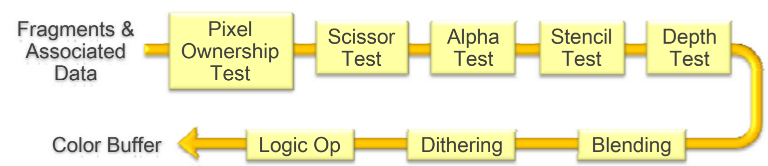
- Determines a final color (RGBA), depth for each fragment
- Generates either 1 or 0 colored fragments (may be discarded)

Stage 5: Raster Operations



Final sequence of per-fragment operations before updating the framebuffer

- fragment may be discarded here as well if any test fails



Graphics Pipeline: Summary

Two programmable stages: vertex and fragment stages

Cg language: high-level language for programming GPUs

- Vertex shaders and fragment shaders
- Operate on streams of data (vertices and fragments)
- Shader is executed repeatedly - once for each element of data in a stream
- Each element of data is independent, in parallel

Content

- ❑ Example Program
- ❑ Graphics Hardware Pipeline
- ❑ Introduction to Cg
- ❑ Cg Programming Examples
  - Multitextures vs. Dependent Textures
  - Render-To-Texture vs. Framebuffer Object (FBO)
  - Multi-pass vs. Single-pass
  - Multiple Render Target (MRT)
  - Early Fragment Kill (Early Z-test)

Cg (C for graphics)

Developed by NVIDIA

Work with both OpenGL and DirectX:

- Application manipulates graphics hardware through 3D API
- Cg compiler: produces the accepted form of code for 3D API and transfers to GPU
- 3D API drivers: perform final translation into hardware-executable code

Cg Runtime:

- core Cg runtime library (cg prefix)
- cgGL and cgD3D libraries

Find Cg Resources online

Latest Cg Toolkit online:

http://www.nvidia.com/object/cg_toolkit.html#cgtutorial

The Cg Tutorial Book and examples online:

http://developer.nvidia.com/object/cg_tutorial_home.html

http://developer.nvidia.com/object/cg_tutorial_software.html

Documents:

Cg Reference Manual

Cg Users Manual

Installing a Cg Program (OpenGL)

Installation

- Download “Cg Install Package” and install it
- In Visual Studio, add new paths for include files and library files in Tools\Options\Projects
- Include files:
  C:\Program Files\NVIDIA Corporation\Cg\include
- Library files:
  C:\Program Files\NVIDIA Corporation\Cg\lib
- Link with cg.lib and cggl.lib
- #include <Cg/cg.h>
  #include <Cg/cgGL.h>

Cg Programming

CPU code:

- Use functions from Cg Runtime
- Prepare and run the Cg program

GPU code:

- Vertex program
- Fragment program

Preparing a Cg Program (OpenGL)

- ❑ Context, profile, program and parameter

First, create a context:

- `CGcontext context = cgCreateContext();`

Define a profile (vertex or fragment):

- `CGprofile profile = cgGLGetLatestProfile(profile_type);`

Create and compile a program:

- `CGprogram program = cgCreateProgram(context, programType, programString, profile, entry_name, args);`

Load a program (pass to the 3D API):

- `cgGLLoadProgram(program);`

Preparing a Cg Program (OpenGL)

Get the handle to a parameter:

- `CGparameter myParameter = cgGetNamedParameter(program, “parameter1");`

Set a parameter before the actual drawing call:

- `cgGLSetParameter3f(myParameter, x, y, z);`

Set sampler parameter:

- `cgGLSetTextureParameter(myParameter, textureName);`

Before and after the actual drawing call:

- `cgGLEnableTextureParameter(myParameter);`
- `cgGLDisableTextureParameter(myParameter);`

Running a Cg Program (OpenGL)

Bind the program:

- `cgGLBindProgram(program);`

Enable the profile:

- `cgEnableProfile(profile);`

After that, the program will execute in subsequent drawing calls

- for each vertex (for vertex programs)
- for each fragment (for fragment programs)
- these programs are often called *shaders*

Running a Cg Program (OpenGL)

Only one vertex / fragment program can be bound at a time

- the same program will execute unless another program is bound

Disable a profile by:

- `cgGLDisableProfile(profile);`

Release resources:

- `cgDestroyProgram(program);`
- `cgDestroyContext(context);`
- the latter destroys all programs as well

Error Handling

There are core Cg routines that retrieve global error variables:

- `error = cgGetError();`
- `cgGetErrorString(error);`
- `cgSetErrorCallback(MyErrorCallback);`

Cg Grammars

Support seven basic data types:

- `float, half, int, fixed, bool, sampler*, string`
- Handle to texture objects: `sampler*: sampler, sampler1D, sampler2D, sampler3D, samplerCUBE, samplerRECT`

Built-in vector data types:

- `float2, float3, float4`
- `half2, half3, half4`
- `fixed2, fixed3, fixed4`

Matrices up to four by four elements:

- `float1x1, float2x2, float3x3, float4x4`

Cg Grammars

Data operations:

Vectors

- Constructor
  e.g. `float4 v = float4(0, 1, 2, 3);`
- Array Operator
  e.g. `v[0], v[1], v[2], or v[3]`
- Swizzle Operator
  `color.rgba, position.xyzw`
  e.g. `v.xyz, v.xxxz, v.yyx, v.yx, v.xyzw`

Matrices

- Constructor

```
float4 v = float4(a,b,c,d);
float4x4 m = float4x4(v,v,v,v);
```


Cg Grammars

Functions:

- Many have direct correspondence to assembly instructions or good approximations
- Linear Algebra Functions
  `dot(a, b)` – Dot Product
  `mul(A, B)` – Matrix-Matrix, Vector-Matrix, or Matrix-Vector multiplication
- Texture Lookup Functions
  - `tex*(sampler* texture, float* texCoord)`
  - * - The dimensionality of the texture
- Geometric Functions
  - `distance, length, normalize, reflect, refract`

Cg Grammars

Semantics:

- `data_type varName : SEMANTIC`
- Binds a Cg program to the rest of the graphics pipeline

```
struct vOutput {
   float4 position   : POSITION;
   float4 color      : COLOR;
  }
```

```
fOutput main(float2 position : POSITION
             float2 texCoord : TEXCOORD0){
         ...
  }
```

- Only used as main function input/output parameters or global variable

Cg Grammars

Type qualifiers:

`uniform`

- Parameter comes from external environment
- The same for each vertex or fragment

`const`

- The same as in C / C++

varying parameters

- Per-vertex or per-fragment varying parameter
- Provided by semantics

Content

SPIE Medical Imaging

- ❑ Example Program
- ❑ Graphics Hardware Pipeline
- ❑ Introduction to Cg
- ❑ Cg Programming Examples
  - Multitextures vs. Dependent Textures
  - Render-To-Texture vs. Framebuffer Object (FBO)
  - Multi-pass vs. Single-pass
  - Multiple Render Target (MRT)
  - Early Fragment Kill (Early Z-test)

1. Multitextures vs Dependent textures

SPIE Medical Imaging

Multitextures

- Same position, multitextures bound

```
float4 main (half3 texUV : TEXCOORD0,
             uniform sampler2D texture0,
             uniform sampler2D texture1 ) : COLOR
{
       float4 tex0 = tex2D(texture0, texUV.xy);
       float4 tex1 = tex2D(texture1, texUV.xy);

       float4 result = lerp(tex0, tex1, texUV.z);

       return result;
}
```


Multitextures vs Dependent textures

SPIE Medical Imaging

Dependent textures

- Texture coordinates are obtained by sampling another texture image

```
float4 main (half3 texUV : TEXCOORD0,
             uniform sampler3D volume_texture,
             uniform sampler1D LUT ) : COLOR
{
       half index = tex3D(volume_texture, texUV);
       float4 result = tex1D(LUT, index);

       return result;
}
```


2. Render-To-Texture vs FBO

SPIE Medical Imaging

Render-To-Texture

Generate intermediate textures

Traditional ways:

- glCopyTexImage2D() and glCopyTexSubImage2D()
- PBuffers bound as a texture

Framebuffer Object (FBO): best solution

- Logical buffers attached: one or more color buffers, one depth buffer and one stencil buffer
- A logical buffer can be attached to one or more FBOs

Render-To-Texture vs FBO

Create a framebuffer object

```
GLuint fbo;
glGenFramebufferEXT(1, &fbo);
```

Create color buffer

```
GLuint color;
glGenTextures(1, &color);
glBindTexture(GL_TEXTURE_2D, color);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height,
     0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
```

Attach it to FBO

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
     GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_2D, color, 0);
```


FBO

Render to color texture

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);
```

Render to the main framebuffer

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```


FBO Example (Edge Detection)

Set up FBO and color buffers

```
GLuint fbo;
GLuint color[3];
glGenFramebuffersEXT(1, &fbo);
glGenTextures(3, color);
for(int i=0; i<3; i++){
        glBindTexture(GL_TEXTURE_RECTANGLE_NV, color[i]);
        glTexImage2D(GL_TEXTURE_RECTANGLE_NV, 0, GL_RGB, pic_w,
            pic_h, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL);
}
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_RECTANGLE_NV, color[0], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
GL_COLOR_ATTACHMENT1_EXT, GL_TEXTURE_RECTANGLE_NV, color[1], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
GL_COLOR_ATTACHMENT2_EXT, GL_TEXTURE_RECTANGLE_NV, color[2], 0);
checkIfFBOisValid();
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
```


3. Multi-pass vs Single-pass

Multi-pass

- Include several intermediate results
- Last pass to output final result
- History reasons: no for-loop support, limited length shader

Single-pass

- Just one pass to output the results

Example: Multi-pass vs Single-pass Edge detection

Multi-pass Edge detection

Example: Edge Detection

- Four rendering passes
- Three color buffers

Pass 1	Pass 2	Pass 3	Pass 4
Generate Gx	Generate Gy	Combine them	Output result
Color[0]	Color[1]	Color[2]	To screen

Multi-pass Edge Detection

Two fragment programs:

1	0	-1
2	0	-2
1	0	-1

...	...	...
...	frag	...
...	...	...

- Mask (for both Gx and Gy)

```
...
for(...)
  sum += mask[x][y] * texRECT(inputTexture, pos).x;
OUT.col.rgba = sum;
...
```

- Combine Gx and Gy

```
...
float x = texRECT(Gx, pos);
float y = texRECT(Gy, pos);
OUT.col = abs(x)+ abs(y);
...
```


Multi-pass Edge Detection

1. Render Gx to color buffer color[0]

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);
glBindTexture(GL_TEXTURE_RECTANGLE_NV, inputTexture);
glEnable(GL_TEXTURE_RECTANGLE_NV);
cgGLBindProgram(fProgram);
cgGLSetMatrixParameterfc(matrixMask, Gx);
cgGLEnableProfile(fProfile);
// render geometry ...
cgGLDisableProfile(fProfile);
glDisable(GL_TEXTURE_RECTANGLE_NV);
```


Multi-pass Edge Detection

2. Render Gy to color buffer color[1]

```
glDrawBuffer(GL_COLOR_ATTACHMENT1_EXT);
glBindTexture(GL_TEXTURE_RECTANGLE_NV, inputTexture);
glEnable(GL_TEXTURE_RECTANGLE_NV);
cgGLBindProgram(fProgram);
cgGLSetMatrixParameterfc(matrixMask, Gy);
cgGLEnableProfile(fProfile);
// render geometry ...
cgGLDisableProfile(fProfile);
glDisable(GL_TEXTURE_RECTANGLE_NV);
```

Multi-pass Edge Detection

3. Render combination to color buffer color[2]

```
glDrawBuffer(GL_COLOR_ATTACHMENT2_EXT);
glActiveTextureARB(GL_TEXTURE0_ARB);
glBindTexture(GL_TEXTURE_RECTANGLE_NV, color[0]);
glEnable(GL_TEXTURE_RECTANGLE_NV);
glActiveTextureARB(GL_TEXTURE1_ARB);
glBindTexture(GL_TEXTURE_RECTANGLE_NV, color[1]);
glEnable(GL_TEXTURE_RECTANGLE_NV);
cgGLBindProgram(fProgramCombine);
cgGLEnableProfile(fProfile);
// render geometry ...
cgGLDisableProfile(fProfile);
glActiveTextureARB(GL_TEXTURE1_ARB);
glDisable(GL_TEXTURE_RECTANGLE_NV);
glActiveTextureARB(GL_TEXTURE0_ARB);
glDisable(GL_TEXTURE_RECTANGLE_NV);
```


Multi-pass Edge Detection

4. Map color[2] onto the screen

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);
glActiveTextureARB(GL_TEXTURE0_ARB);
glBindTexture(GL_TEXTURE_RECTANGLE_NV, color[2]);
glEnable(GL_TEXTURE_RECTANGLE_NV);
// render geometry ...
glDisable(GL_TEXTURE_RECTANGLE_NV);
```


Single-pass Edge detection

Single-pass

- Only one pass to output final result

  Generate Gx, Gy
  Combine them
  Output result

- No need for intermediate color buffers

Single-pass Edge Detection

Fragment Code:

```
fragout main( float4 TexCoords : TEXCOORD0,
        uniform samplerRECT   inputTexture   : TEXUNIT0,
        uniform float3x3      maskX,
        uniform float3x3      maskY)
{
        fragout OUT;
        ...
        for (...){
            c = texRECT(inputTexture, ...).x;
            sumx += maskX[x][y] * c;
            sumy += maskY[x][y] * c;
        }
        OUT.col.rgba = abs(sumx) + abs(sumy);
        return OUT;
}
```

Single-pass Edge Detection

SPIE Medical Imaging

CPU code: No FBO applied

```
glBindTexture(GL_TEXTURE_RECTANGLE_NV, inputTexture);
glEnable(GL_TEXTURE_RECTANGLE_NV);
cgGLBindProgram(fProgram);
cgGLSetMatrixParameterfc(matrixMask, Gx);
cgGLSetMatrixParameterfc(matrixMask1, Gy);

cgGLEnableProfile(fProfile);
// render geometry ...
cgGLDisableProfile(fProfile);
glDisable(GL_TEXTURE_RECTANGLE_NV);
```


4. MRT (Multiple Render Target)

SPIE Medical Imaging

Allow fragment shader to output multiple color values at one time and write them into separate off-screen render targets of the same resolution

Require extension GL_ARB_draw_buffers

FBOs provide a flexible interface

Example: modified multi-pass Edge Detection

combine first two passes together

MRT (Multiple Render Target)

SPIE Medical Imaging

Set two render targets:

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
    GL_COLOR_ATTACHMENT0_EXT, GL_TEXTURE_RECTANGLE_NV, color[0], 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
    GL_COLOR_ATTACHMENT1_EXT, GL_TEXTURE_RECTANGLE_NV, color[1], 0);


GLuint drawbuffers[2] = {GL_COLOR_ATTACHMENT0_EXT,
                         GL_COLOR_ATTACHMENT1_EXT};
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glDrawBuffers(2, drawbuffers);

// Render the Geometry here ...
```


MRT (Multiple Render Target)

SPIE Medical Imaging

Fragment Code:

```
struct mrt_output {
       float4 color0 : COLOR0;
       float4 color1 : COLOR1;
};
mrt_output main(...)
{
       mrt_output outColor;
       ...
       outColor.color0.rgba = ...;
       outColor.color1.rgba = ...;
       return outColor;
}
```

5. Early Fragment Kill

SPIE Medical Imaging

Tests Z values of pixels before entering the fragment shading pipeline

Much useless work avoided, improving performance and conserving power

Two passes:

- First pass: values written to a depth mask, some pixels are “masked out”
- Second pass: computational expensive shader only processes pixels not masked out

Early Fragment Kill

SPIE Medical Imaging

```
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LEQUAL);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glDisable(GL_BLEND);
glDisable(GL_ALPHA_TEST);
glDisable(GL_SCISSOR_TEST);
glDisable(GL_STENCIL_TEST);
glDepthMask(GL_TRUE);
glDrawBuffer(GL_BACK);
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
// Render geometry here... with no active shader
glDepthMask(GL_FALSE);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
// Render the same geometry here, with active shader
glDisable(GL_DEPTH_TEST);
glEnable(GL_BLEND);
...
```

Enable depth test

Depth comparison mode

Disable effects of all other tests

Enable writing to depth buffer

Disable write to color buffer

Disable depth buffer writing & enable color buffer writing

reset everything

Conclusion

SPIE Medical Imaging

- Example Program
- Graphics Hardware Pipeline
- Introduction to Cg
- Cg Programming Examples
  - Multitextures vs. Dependent Textures
  - Render-To-Texture vs. Framebuffer Object (FBO)
  - Multi-pass vs. Single-pass
  - Multiple Render Target (MRT)
  - Early Fragment Kill (Early Z-test)

Course Schedule

SPIE Medical Imaging

1:30 – 2:00:	Introduction (Klaus Mueller)
2:00 – 2:45:	Graphics-style GPU programming with CG (Wei Xu)
2:45 – 3:00:	GPGPU-style GPU programming with CUDA (Ziyi Zheng)
	Coffee Break
3:30 – 4:00:	GPGPU-style GPU programming with CUDA (Ziyi Zheng)
4:00 – 4:20:	CT reconstruction pipeline components (Klaus Mueller)
4:20 – 5:20:	GPU-accelerated CT reconstruction (Fang Xu)
5:20 – 5:30:	Extensions and final remarks (all)