

## Lecture 10 of 41

# Introduction to DirectX & Direct3D


## Lab 2a: Direct3D Basics

**William H. Hsu**  
 Department of Computing and Information Sciences, KSU

KSOL course pages: <http://bit.ly/hGvXIH/> / <http://bit.ly/eVizrE>  
 Public mirror web site: <http://www.kddresearch.org/Courses/CIS636>  
 Instructor home page: <http://www.cis.ksu.edu/~bhsu>

Readings:  
 Today: Section 2.7, Eberly 2<sup>e</sup> – see <http://bit.ly/ieUq45>: [Direct3D handout](#)  
 Handout based on material © 2004 – 2010 K. Ditchburn: <http://bit.ly/hMqXMI>  
 Next class: Sections 2.6.3, 20.3 – 20.4, Eberly 2<sup>e</sup>  
 Wikipedia article: <http://en.wikipedia.org/wiki/Shader>


CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



## Lecture Outline

- Reading for Last Class: §2.5, 2.6.1 – 2.6.2, 4.3.2, 20.2, Eberly 2<sup>e</sup>
- Reading for Today: §2.7, Eberly 2<sup>e</sup>; [Direct3D handout](#)
- Reading for Next Class: §2.6.3, 20.3 – 20.4, Eberly 2<sup>e</sup>
- Last Time: Intro to Illumination and Shading
  - \* Local vs. global models
  - \* Illumination (vertex shaders) vs. shading (fragment/pixel shaders)
  - \* Phong illumination equation: introduction to shading
- Previously: Projections, Modelview, Viewing, Clipping & Culling
- Today: Direct3D Design Overview, Programming Intro
- Hardware Rendering: Application Programmer Interfaces (APIs)
- Shaders in Modern Pipeline
  - \* **Vertex shaders**: vertex attributes to illumination at vertices
  - \* **Pixel shaders**: lit vertices to pixel colors, transparency
- Next: Texture Mapping Explained; Shading in OpenGL


CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



## Vertex Shaders vs. Pixel Shaders

- Classical **Fixed-Function Pipeline (FFP)**: Per-Vertex Lighting, MVT + VT
  - \* Largely superseded on desktop by programmable pipeline
  - \* Still used in mobile computing
- Modern **Programmable Pipeline**: Per-Pixel Lighting
  - \* **Vertex Shaders** (FFP and Programmable)
    - \* Input: per-vertex attributes (e.g., object space position, normal)
    - \* Output: lighting model terms (e.g., diffuse, specular, etc.)
  - \* **Pixel Shaders** (Programmable Only)
    - \* Input: output of vertex shaders (lighting aka illumination)
    - \* Output: pixel color, transparency (R, G, B, A)
- Brief Digression
  - \* Note: vertices are *lit*, pixels are *shaded*
    - > “Pixel shader”: well-defined (iff “pixel” is)
    - > “Vertex shader”: misnomer (somewhat)
  - \* Most people refer to both as “shaders”

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



## Where We Are

Lecture	Topic	Primary Source(s)
0	Course Overview	Chapter 1, Eberly 2 <sup>e</sup>
1	CG Basics: Transformation Matrices; Lab 0	Sections §1.2.1, 2.2
2	Viewing 1: Overview, Projections	§ 2.3 – 2.2.4, 2.6
3	Viewing 2: Viewing Transformation	§ 2.3 esp. 2.3.4, FV/FH slides
4	Lab 1a: Flash & OpenGL Basics	Ch. 4, 16 <sup>e</sup> , <a href="#">Angel Primer</a>
5	Viewing 3: Graphics Pipeline	§ 2.3 esp. 2.3.7, 2.6, 2.7
6	Scan Conversion 1: Lines; Midpoint Algorithm	§ 2.5.1, 3.1 FV/FH slides
7	Viewing 4: Clipping & Culling; Lab 1b	§ 2.3.5, 2.4, 3.1.3
8	Scan Conversion 2: Polygons; Clipping Intro	§ 2.4, 2.5 esp. 2.5.4, 3.1.6
9	Surface Detail 1: Illumination & Shading	§ 2.5, 2.6, 1 – 2.6.2, 4.3.2, 20.2
10	Lab 2a: Direct3D: Direct3D Intro	§ 2.7, <a href="#">Direct3D handout</a>
11	Surface Detail 2: Textures; OpenGL Shading	§ 2.6.3, 20.3 – 20.4, <a href="#">Primer</a>
12	Surface Detail 3: Mappings; OpenGL Textures	§ 20.5 – 20.13
13	Surface Detail 4: Pixel/Vertex Shad.; Lab 2b	§ 3.1
14	Surface Detail 5: Direct3D Shading; CG/SL	§ 3.2 – 3.4, <a href="#">Direct3D handout</a>
15	Demos 1: CGA, Fun, Scene Graphs; State	§ 4.1 – 4.3, <a href="#">CGA handout</a>
16	Lab 3a: Shading & Transparency	§ 2.8, 20.1, <a href="#">Primer</a>
17	Animation 1: Basics, Keyframes; HW/Exam	§ 5.1 – 5.2
18	Exam 1 review: Hour Exam 1 (evening)	Chapters 1 – 4, 20
18	Scene Graphs: Rendering; Lab 3b: Shader	§ 4.4 – 4.7
19	Demos 2: SPX; Skinning, Morphing	§ 5.3 – 5.6, <a href="#">CGA handout</a>
20	Demos 3: Surfaces; B-reps/Volume Graphics	§ 10.4, 12.7, <a href="#">Mesh handout</a>

Lightly-shaded entries denote the due date of a written problem set; heavily-shaded entries, that of a machine problem (programming assignment); blue-shaded entries, that of a paper review, and the green-shaded entry, that of the term project.  
 Green, blue and red letters denote exam review, exam, and exam solution review dates.

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



## Acknowledgements



**Nathan H. Bean**  
 Instructor  
 Outreach Coordinator  
 Department of Computing and Information Sciences  
 Kansas State University  
<http://bit.ly/gC3vwt/>

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwt/>





**Andy van Dam**  
 T. J. Watson University Professor of  
 Technology and Education & Professor of  
 Computer Science  
 Brown University  
<http://www.cs.brown.edu/~avdl/>



**Randy Fernando**  
 Executive Director  
 Mindful Schools  
<http://www.randima.com>



**Mark Kilgard**  
 Principal System Software Engineer  
 Nvidia  
<http://bit.ly/gdjlzR>

Cg material from figures © 2003 R. Fernando & M. Kilgard, Nvidia, from *The Cg Tutorial*  
<http://bit.ly/S9MSR>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



## Direct3D

- Popular 3-D Graphics Library of DirectX (Sometimes Called “DirectX”)
- \* Top market share due to Windows & Xbox platforms
- \* Many developers (has overtaken OpenGL except on smartphones)
- Can Implement Many Effects We Have Studied/Will Study Such As:
  - \* Cube map (Lecture 12)
  - \* Particle system (Lecture 28)
  - \* Instancing (Lecture 20 & Advanced CG)
  - \* Shadow volume (Lecture 12)
  - \* Displacement Mapping (Lecture 12)







© 2006 D. Blythe, “The Direct3D® 10 System”.  
<http://bit.ly/133uMv>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

7

## Preview: Shader Languages

OpenGL State Machine (Simplified) from Wikipedia: Shader  
<http://bit.ly/8B8BP>

Remember this from Lecture 04?

- **HLSL**: Shader language and API developed by **Microsoft**, only usable from within a DirectX application.
- **Cg**: Shader language and API developed by **Nvidia**, usable from within a DirectX and OpenGL application. Cg has a stark resemblance to HLSL.
- **GLSL**: Shader language and API developed by the **OpenGL** consortium and usable from within an OpenGL application.

© 2009 Koen Samyn  
<http://knol.google.com/k/hisl-shaders>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

8

## DirectX

- DirectX is a collection of APIs developed by Microsoft for multimedia and game applications
- They utilize the *Component Object Model* (COM), allowing you to use them across a wide variety of languages
- The entire DirectX API is available freely from Microsoft, and works with any current Visual Studio compiler (and many others)
- Today we'll be focusing on the API for 3D rendering, Direct3D

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3wvH>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

9

## Abstraction

- The primary purpose of Direct3D is to provide a uniform interface for working directly with the graphics hardware without having to go through windows.
- Direct3D applications actually render to a display surface within the confines of a window – they don't use the Window API
- For best performance, you want to use full-screen, exclusive mode. In this mode, Windows does **no rendering whatsoever**, allowing you to leverage the full resources of the computer

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3wvH>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

10

## Rendering Basics

- Like OpenGL, Direct3D uses vertex-based polygon mesh data to render 3D Data
- All 3D graphics are composed of triangles, usually these triangles share sides in what we refer to as a triangle mesh, or mesh.
- Each triangle is defined by its three corners, which we refer to as vertices
- We can store more data on vertices than just their position in 3D space – texture coordinates, normal vectors, and ambient colors for example

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3wvH>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

11

## Direct3D Device

- Direct 3D places most of its rendering control in a Direct3D Device object.
- Essentially, the Direct3D Device is a wrapper around the graphics hardware, and exposes most of its functionality to the programmer through a common interface.
- You can poll the Direct3D Device to learn of the graphics hardware's capabilities

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3wvH>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

12

## Direct3D Rendering Pipeline

- The process of transforming model geometry into on-screen images is called the rendering pipeline
- In DirectX, the pipeline runs *entirely on the graphics hardware* – **there is no software fallback**
- Other than that difference, it shares many characteristics with other 3D rendering libraries

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3wvH>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

15

## Input Assembly

- Input is sent from the DirectX application in the form of Vertex data (both buffered and unbuffered), shader code and variables, and textures
- When possible, these sources are stored in the on-board graphics card memory

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

14

## Vertex Formats [1]: Flexible Vertex Format (FVF)

- You also will define the format of vertices that you will send to the graphics card.
- We use the Flexible Vertex Format (FVF) to do this
- FVF is a two part process – we define a struct representing our vertex format, and define a constant describing the structure of the format.

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

15

## Vertex Formats [2]: Using FVF

```
#define POSITION_NORMAL_TEXTURE_FVF(D3DFVF_XYZ |
D3DFVF_NORMAL | D3DFVF_TEX1)

struct PositionNormalTextureVertex
{
    float x,y,z;
    float nx, ny, nz;
    float tu, tv;
};
```

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

14

## Vertex Formats [3]: For More Details on FVF

- Flexible Vertex Formats allow us to send the bare minimum of vertex data for our needs to the graphics card.
- Since the pipeline from our CPU to the graphics hardware tends to be a bottleneck, this is a **must** when speed is a concern.
- More detail on what flags exist for the DirectX FVF can be found in the DirectX API documentation

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

17

## FVF Graphically Expressed

Graphics courtesy of <http://www.directxtutorial.com>

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

18

## Vertex Buffers

- Vertex Buffers are used to hold vertex data and to stream this data to the graphics card
- Buffers may be located in Video memory or in RAM, depending on your device setup
- Buffers are a shared resource – both the CPU and the GPU will be accessing them at times. Because of this, we will need to lock the buffer before we place our data into it.
- To minimize overhead, it is common practice to create your vertex data in CPU-only memory, then do a fast copy (memcpy) into a locked Vertex buffer

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



## Rendering Vertex Buffers

- First, you need to set it as a stream source on your Direct3D Device
- Then you can call `DrawPrimitive` on your device, indicating what portion of the Vertex Buffer should be drawn, along with what kind of primitive you are rendering
- This starts the vertex data on the Rendering Pipeline

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>



## Index Buffers

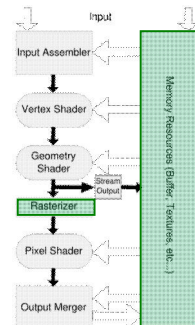
- Index Buffers are an array of indices which reference an associated Vertex Buffer
- Index Buffers allow us to re-use Vertex data elements when creating meshes – this is especially useful because triangles in most meshes are sharing their vertices with adjacent triangles.
- By using an index buffer, we only have to define a vertex once, then we can define the triangle by indices; and since an index is usually quite a bit smaller than a vertex's data, we send less data into our pipeline.

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>



## Rasterization

- **Rasterization**
  - \* process by which geometry is converted into pixels for rendering
  - \* much the same with any rendering process
- In general, refers to all vector-to-raster stages
  - \* transformations
  - \* clipping
  - \* scan conversion
- Sometimes equated with scan conversion (final step)
- Direct3D maintains useful buffers for it
  - \* depth buffers
  - \* stencil buffers

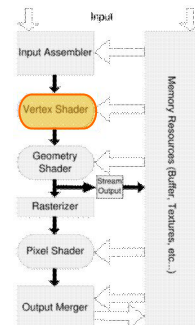


Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>



## Vertex Shaders [1]

- The next step down the pipeline are the Vertex Shaders
- Vertex shaders allow us to manipulate our vertex data on the graphics card
- This is important, because it means we'll be using the GPU



Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>



## Vertex Shaders [2]

- There are a number of reasons we want to manipulate our vertex data on the GPU rather than the CPU:
  - \* The GPU is *highly optimized* for Vector- and Matrix-based operations. The CPU is not.
  - \* Because these operations operate independently from the CPU, our CPU is now free for other operations
- Vertex Shaders are essentially little programs that run against every vertex in the data we've sent down the rendering pipeline
- We write vertex shaders using Microsoft's High-Level Shader Language


Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>



## High-Level Shader Language [1]

- HLSL was originally developed as part of a joint partnership between Nvidia and Microsoft, which also was responsible for Cg
- It sports a very C-like syntax, but also supports arbitrary sized vectors and matrices
  - \* e.g., 4x4 matrix of floats can be defined by `float4x4`
  - \* 3-element vector of floats can be defined by `float3`
- We can also pass variables into our shaders from DirectX as externs

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>


25 

## High-Level Shader Language [2]

- HLSL also sports two vector component namespaces, which provide helpful shorthand when dealing with positions and colors: xyzw and rgba
  - e.g., color.r is equivalent to color[0]
  - position.y is equivalent to position[1]
- We can also *swizzle* our vectors – access more than one component in a single call
  - e.g., color.gb += 0.5f; adds 0.5 to both the green and blue components of the vector color
- Finally, HLSL provides a lot of useful functions – you can read the API documentation for more

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University


26 

## Uses for Vertex Shaders

- Vertex shaders are primarily used to transform our vertices. Some Examples:
  - Dynamic deformation of models
  - Displacement Mapping
  - Billboards
  - Fluid modeling
  - Particles
- Vertex shaders can be further combined with other shader types for a wide variety of possible effects

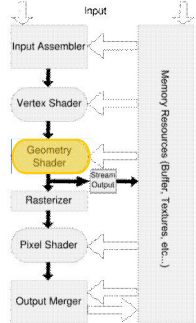
Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

27 


## Geometry Shaders [1]: Implementation

- Geometry Shaders are a new feature for DirectX 10
- They aren't supported by the Xbox 360, and most computers at the moment
- But they offer some exciting possibilities for the near future



Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University


28 

## Geometry Shaders [2]: Uses

- The primary purpose of a Geometry Shader is to *create new geometry* using the vector data supplied from the Vector Shader
- This means they can be used for
  - Generating point sprites
  - Extruding shadow volumes (to use in the pixel shader)
  - Further tessellation operations – i.e. increasing the level of tessellation in models (more triangles == smoother models)
  - Subdivision surfaces (e.g., terrain)
  - Single-pass cube map generation

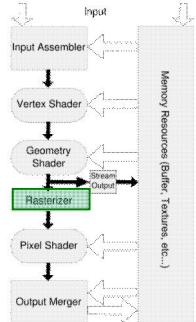
Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

29 


## Rasterization Revisited: Per-Pixel Operations

- Recall: Definition**
  - Conversion of geometry into pixels for rendering
  - Common/similar step across rendering pipelines
- Input Assembler Takes Care of Viewing Transformation (VT)**
- Per-Vertex Operations**
  - Clipping
  - Culling
  - Lighting (for flat or Gouraud shading; not Phong shading)
- Per-Pixel Operations**
  - Scan conversion of primitives
  - Pixel shading



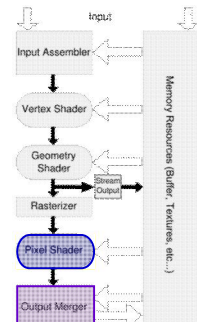
Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

30 


## Pixel Shaders [1]: Implementation

- Pixel Shaders allow further transformation on the rasterized pixel data
- Like Vertex Shaders, Pixel Shaders are written in HLSL, use the same syntax and function libraries, and can have variables set by Direct3D



Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwh>


CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

31  **Pixel Shaders [2]: Uses**

- Pixel Shaders can be used in a number of ways:
  - To control how multiple textures are blended when texturing a surface
  - To alter the color of a material
  - For customized lighting
  - To apply specular highlights

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwH>


CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

32  **"Managed" DirectX**

- Direct3D offers the programmer a lot of control, but careful attention must be paid to memory and resource management
- Microsoft piloted a "Managed" DirectX effort at one point. This, combined with the library they developed for the original X-Box, developed into XNA, a shared library, framework, and IDE for building game applications in C#

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University  
<http://bit.ly/gC3vwH>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

33  **Drawing in Direct3D [1]: Defining Flexible Vertex Formats**

- Position & Normal Only vs. Position & Color Only**

```

struct CUSTOMVERTEX { D3DXVECTOR3 p; // Vertex position
                      D3DXVECTOR3 n; // Vertex normal
};
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_NORMAL)

struct MYVERTEX { D3DXVECTOR3 p; // Vertex position
                 DWORD colour;
};
#define MYVERTEX_FVF (D3DFVF_XYZ | D3DFVF_DIFFUSE)
  
```

- With Textures**

```

struct CUSTOMVERTEX { D3DXVECTOR3 p; // Vertex position
                      D3DXVECTOR3 n; // Vertex normal
                      float tu,tv; // texture co-ordinate
};
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1)


struct CUSTOMVERTEX { float x,y,z; // vertex position
                      float nx,ny,nz; // vertex normal
                      float tu,tv; // texture co-ordinate
};
  
```

- Components**

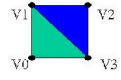
D3DFVF_XYZ	- position in 3-D space
D3DFVF_XYZRHW	- already transformed co-ordinate (2-D space)
D3DFVF_NORMAL	- normal
D3DFVF_DIFFUSE	- diffuse colour
D3DFVF_SPECULAR	- specular colour
D3DFVF_TEX1	- one texture co-ordinate
D3DFVF_TEX2	- two texture co-ordinates

Toymaker © 2004 – 2010 K. Ditchburn, Teesside University  
<http://bit.ly/hMaxMI>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

34  **Drawing in Direct3D [2]: Steps & Example**

- Specify the material we wish to use for the following triangles
- Specify the texture we wish to use (if we want one or NULL if not)
- Set the stream source to our vertex buffer
- Set the FVF we will be using
- Set the index buffer we will be using
- Call the required `DrawPrimitive` function




```

void CGfxEntityCube::Render()
{
    gD3DDevice->SetMaterial( &m_material );
    gD3DDevice->SetTexture(0,NULL);
    gD3DDevice->SetStreamSource( 0, m_vb,0, sizeof(CUBEVERTEX) );
    gD3DDevice->SetFVF( D3DFVF_CUBEVERTEX );
    gD3DDevice->SetIndices( m_ib);

    // draw a triangle list using 24 vertices and 12 triangles
    gD3DDevice->DrawIndexedPrimitive( D3DPT_TRIANGLELIST,0,0,24,0,12);
}
  
```

Toymaker © 2004 – 2010 K. Ditchburn, Teesside University  
<http://bit.ly/hMaxMI>


CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

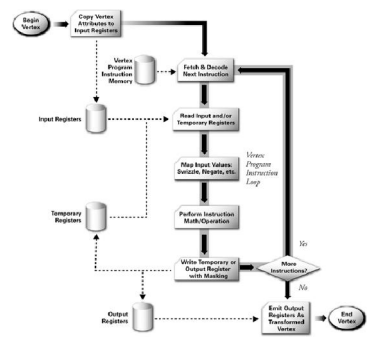
35  **Direct3D vs. OpenGL**

- Most people find OpenGL easier to learn to start with
- OpenGL uses standard C interfaces.
- Direct3D has a steeper initial learning curve and is based on C++ interfaces (COM).
- It is more difficult using OpenGL to do lower level operations than Direct3D, however this does mean you are less likely to crash an OpenGL app than a Direct3D one.
- The vast majority of PC games are written using Direct3D.
- OpenGL can run on multiple platforms where as Direct3D is Windows based.
- Direct3D is updated frequently (every 2 months), allowing a standard interface to new features sooner than they appear in core OpenGL. However, OpenGL has an extension mechanism allowing immediate access to new features as graphics manufacturers create them, rather than having to wait for a new API version to appear.
- Direct3D has a lot of helper functions for common game coding issues (the D3DX set of functions)
- In terms of performance there is little difference between the two.

Toymaker © 2004 – 2010 K. Ditchburn, Teesside University  
<http://bit.ly/hMaxMI>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

36  **Preview: Vertex Shaders**



© 2003 R. Fernando & M. Kilgard. The Cg Tutorial.  
<http://bit.ly/59HRSR>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University



37

## Preview: Pixel & Fragment Shaders

© 2003 R. Fernando & M. Kågard. *The Cg Tutorial*.  
<http://bit.ly/59f5RSR>

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

38

## Next Time: Texture Mapping!

- Idea: enhance visual appearance of plain surfaces by applying fine structured details

Eduard Gröller, Stefan Jeschke  
 Adapted from slides  
 © 2002 Gröller, E. & Jeschke, S. Vienna Institute of Technology

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

39

## Summary

- Last Time: Intro to Illumination and Shading
  - Local vs. global models
  - Illumination (vertex shaders) vs. shading (fragment/pixel shaders)
  - Phong illumination equation: introduction to shading
- Previously: Classical Fixed-Function Pipeline
- Today: Shaders in Modern Pipeline
  - Vertex shaders
    - Input: per-vertex attributes (e.g., object space position, normal)
    - Output: lighting model terms (e.g., diffuse, specular, etc.)
  - Pixel shaders
    - Input: output of vertex shaders (lighting aka illumination)
    - Output: pixel color, transparency (R, G, B, A)
- Hardware Rendering: Application Programmer Interfaces (APIs)
- Drawing in Direct3D: See *Toymaker Site* – <http://bit.ly/hMqxMI>
- Next: Shader Languages – (O)GLSL, HLSL / Direct3D, Renderman

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University

40

## Terminology

- Direct3D**: Graphics Component of Microsoft *DirectX* Library
  - Similarities to OpenGL: C/C++, 3-D polygons-to-pixels pipeline
  - Differences vs. OpenGL: Windows, COM, some lower-level ops
- Classical Fixed-Function Pipeline (FFP): Per-Vertex Lighting, MVT + VT
- Modern Programmable Pipeline: Per-Pixel Lighting
- Vertex Shaders (FFP and Programmable)
  - Input: per-vertex attributes (e.g., object space position, normal)
  - Output: lighting model terms (e.g., diffuse, specular, etc.)
- Pixel Shaders (Programmable Only)
  - Input: output of vertex shaders (lighting aka illumination)
  - Output: pixel color, transparency (R, G, B, A)
- Shader Languages
  - Domain-specific programming languages
  - Geared towards hardware rendering
  - (O)GLSL, HLSL / Direct3D, Renderman

CIS 536/636 Introduction to Computer Graphics Lecture 10 of 41 Computing & Information Sciences Kansas State University