
By Raphael Mun
Quick Review...

- Graphics Pipeline
  1. Set State Variables
  2. Send Vertices
  3. Generate Image
- From the CPU -> GPU
Process of Abstraction

- Basis of Graphics
  - Set Transforms
  - Set Vertices
    - Floating Points multiplied and added to each other
- Rules applied on top of this
  - Transforms multiply to each other
  - Vertices & Vectors multiply to a Transform
- Sets of 3 Vertices create a triangle
  - Each Vertex has a corresponding set of more floats
    - Normal Vector
    - Color [Amb, Dif, Spc, Em, ...]
    - Texture
COMPUTER

- **Assembly**
  - Direct Translation of Machine Code
  - (Add, Subtract, Multiply Divide)
- **Functions**
  - Input-output operation (variable -> value)
- **Data Structures**
  - Sets of Variables
- **Object-Orientation**
  - Sets of data w/ set of functions
- **Application Programming Interface** (API)

GRAPHICS

- **Floating point arithmetic**
  - Direct translation of graphics processing
  - (Add, Subtract, DotProd, CrossProd)
- **Transforms**
  - Input-output operation (floats -> color)
- **"Vectors"/"Vertices"**
  - Sets of Floats
- **Vertex -> Polygon**
  - Data
    - Position Vertex
    - Normal Vector
    - Color Vectors
    - Texture Coordinates
  - Functions
    - Vertex Transformation
    - Barycentric Color interpolation
    - Lighting Calculation
    - Depth Checking
    - ...
- **Graphics Interface**
  - OpenGL, DirectX, ...
Where to draw the line between CPU/GPU?

Beneath the hood, Graphics APIs do:

1. **Vertex Input**
2. **Vertex Transformation**
   - MODELVIEW -> (Lighting, ...) -> PROJECTION
3. **Internal Processing**
   - Back-face Culling
   - Depth Check
   - Clipping
   - ...
4. **Pixel Transformation**
   - [Polygons] TEXTURE -> (Color Interpolation) -> Monitor
   - [Lines, Points] Rasterize -> Monitor

**Spaces**:
- Local-space
- World-space
- Screen-space
- Pixel-space
Where to draw the line between CPU/GPU? (Cont’d)

- Vertex Shaders (single-vertex input)
  - Vertex Transformation Replacement
  - Can be done in CPU/GPU
  - Replaces MODELVIEW, PROJECTION...

- Pixel Shaders (single-pixel input)
  - Pixel Transformation Replacement
  - Done in GPU
  - Replaces TEXTURE Coordinates, Pixel Color, Bump Mapping, ...
Fixed-Function Pipeline Limitations

- Large, Clunky
  - Work with big buffers at a time
- Inflexible
- Made to simply chug out triangles
- No scene information, single raycast
As the GPU gains more processing power it can...

- Work with separate triangles
- Work with individual pixels
- Start gaining scene information

Replacing the pipeline

Pixel Shaders (pixel) → Vertex Shaders (vertex) → [Geometry Shaders] (models/triangles)
Shader Flexibility – Biggest Advantage/Drawback

- Interpret & manipulate vertex/matrices as non-coordinate data
- Full understanding of graphics, lighting, etc. required

Unable to create data
- 1 Input -> 1 Output
I = A_{intensity} * A_{color} + D_{intensity} * D_{color} * N.L + Specular \quad (A = \text{Ambient}, \, D = \text{Diffuse})

Ignoring Specular, just Ambient + Diffuse Per-Pixel Lighting

float4x4 matWorldViewProj;
float4x4 matWorld;
float4 vecLightDir;

struct VS_OUTPUT
{
  float4 Pos : POSITION;
  float3 Light : TEXCOORD0;
  float3 Norm : TEXCOORD1;
};

VS_OUTPUT VS(float4 Pos : POSITION, float3 Normal : NORMAL)
{
  VS_OUTPUT Out = (VS_OUTPUT)0;
  Out.Pos = mul(Pos, matWorldViewProj); // transform Position
  Out.Light = vecLightDir; // output light vector
  Out.Norm = normalize(mul(Normal, matWorld)); // transform Normal and normalize it
  return Out;
}

float4 PS(float3 Light: TEXCOORD0, float3 Norm : TEXCOORD1) : COLOR
{
  float4 diffuse = { 1.0f, 0.0f, 0.0f, 1.0f};
  float4 ambient = {0.1, 0.0, 0.0, 1.0};
  return ambient + diffuse * saturate(dot(Light, Norm));
}

http://www.gamasutra.com/features/20030418/engel_01.shtml
Future of Graphics Hardware

- Shaders -> Scene Data
  - Shared System/Video Memory (Xbox 360)
  - Geometry Shaders
  - ‘Instancing’
- Multi-core Graphics Cards
  - Fast floating-point processing
  - Physics, AI, etc.
Questions?