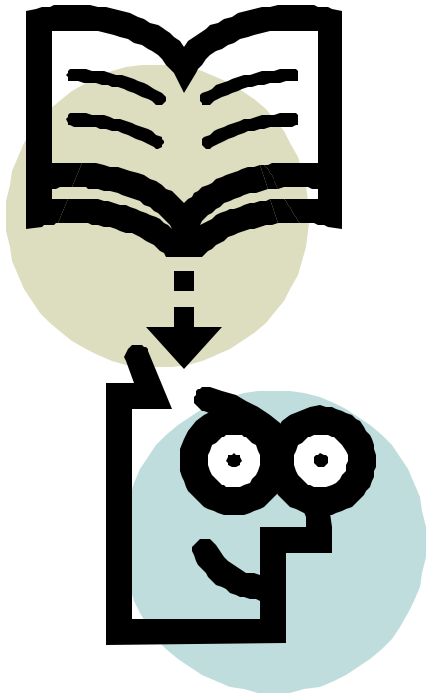



CS 563 Advanced Topics in Computer Graphics
Shader Programming with Cg

by Michael Schmidt

Outline

- History of Shaders
- Current Languages
- Introduction to Cg
- A Smathering of Math
- Example Program
- Cg Setup
- Textures
- Fun with Light



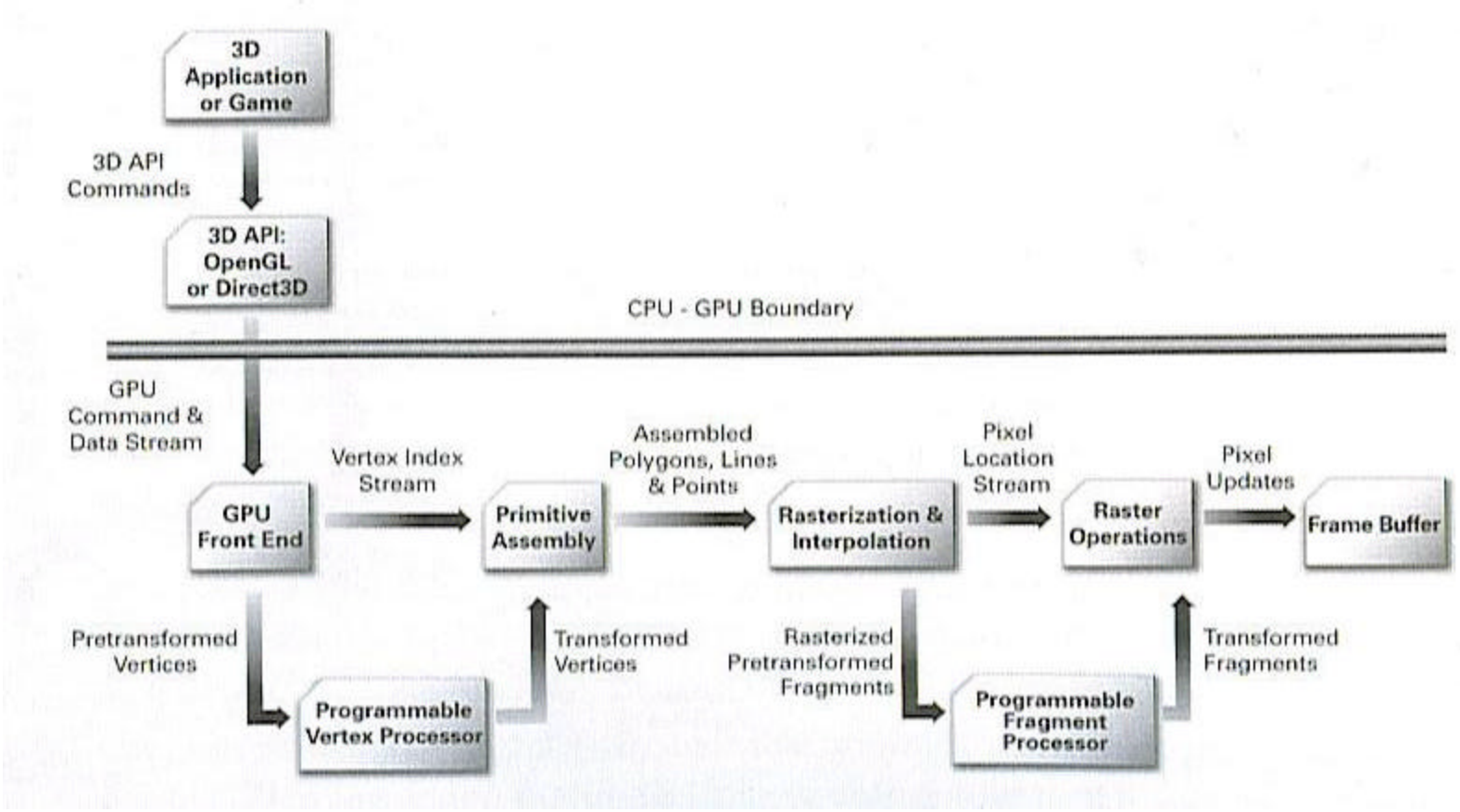
History of Shaders

- RenderMan - offline.
- PixelFlow and RealTime Shading Language – real-time.
- Current languages are their descendants.
- programmable GPUs = real-time shaders.
- Shaders can be written in assembly, but there are huge benefits to using a high level language.
- Four classes of operations: constant for a scene, constant for an object, vary per vertex, vary per pixel.

Current Languages

- GLSL – GL Shading Language – Connected to OpenGL.
- HLSL – High Level Shading Language – Microsoft language for use with Direct3D.
- Cg – Nvidia language for use with OpenGL or Direct3D.

Vertex and Fragment Programs



From “The Cg Tutorial”, pg. 17

Cg Design

- Cg is a shading language based on the syntax of C.
- Cg exists to take advantage of the specialized design of GPUs.
- Cg is used along with traditional general purpose languages.
- Cg is robust and extensible.
- The Cg runtime manages Cg programs.
- Cg programs work with either OpenGL or Direct3D.
- Multiple Cg programs per application.

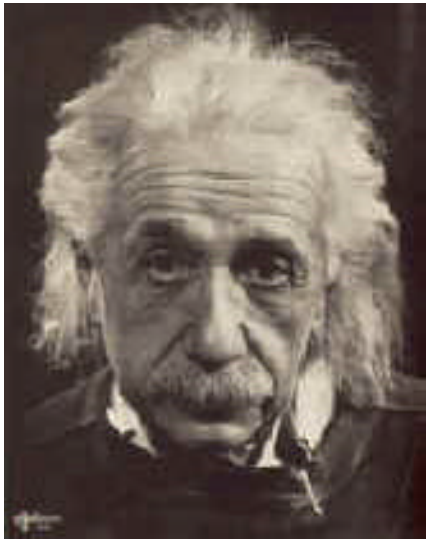
Unique Cg Syntax

- Cg supports new special “packed” types such as float4 and float4x4.
- The GPU performs operations on packed types efficiently.
- Prefer packed types over arrays.
- Semantics tell the graphics hardware how an identifier should be used.
- Functions can be entry functions or internal functions.
- “Uniform” keyword specifies external origin.

More Uniqueness

- “out” parameters are initially undefined, but they must be set before the function returns.
- This is called “call-by-result” and it is different than “call-by-reference”.
- “in” parameters are called by value.
- “inout” parameters combine both ideas.

Math is Good



- The operators * / - + all work on scalars and vectors.
- Logical operators are supported and bitwise operators are reserved.
- The half type offers precision between float and double.
- The fixed data type has a range from -2 to almost 2, but is available only in fp30 and up.

Swizzling

- Besides being fun to say, swizzling can be a powerful tool.
- Swizzling rearranges vector components.
- RGBA or XYZW values can be swizzled, but not mixed.
- Swizzling examples:
  - float3 vec = scalar.xxx;
  - float4 vec1 = vec2.bgra;
- Parts of vectors can be assigned new values.
- Write Masking example:
  - vec3.xy = vec4;

Standard Library

- Cg provides Standard Library routines that have been optimized to run quickly on a GPU.

abs	isnan	radians	texCUBE	refract
cos	lerp	reflect	sin	smoothstep
cross	log2	round	sincos	
determinant	max	rsqrt	normalize	
dot	mul	tex2D	pow	
floor	pow	tex3Dproj	reflect	

Flow Control

- If, else, for, while, and do while are all implemented in Cg.
- Some profiles support loops only if they can determine the number of iterations.
- Dynamic loops are available on NVidia's fourth generation hardware (NV30 and up).
- Maximum vertex program size is 65,536 and maximum fragment program size is 1,024 instructions.

Cg Keywords

asm*	explicit	pixelfragment*	template
asm_fragment	extern	pixelshader*	texture*
auto	FALSE	private	texture1D
bool	fixed	protected	texture2D
break	float*	public	texture3D
case	for	register	textureCUBE
catch	friend	reinterpret_cast	textureRECT
char	get	return	this
class	goto	row_major	throw
column_major	half	sampler	TRUE

Cg Keywords

compile	if	sampler_state	try
const	in	sampler1D	typedef
const_cast	inline	sampler2D	typeid
continue	inout	sampler3D	typename
decl*	int	samplerCUBE	uniform
default	interface	shared	union
delete	long	short	unsigned
discard*	matrix	signed	using
do	mutable	sizeof	vector*
double	namespace	static	vertexfragment*

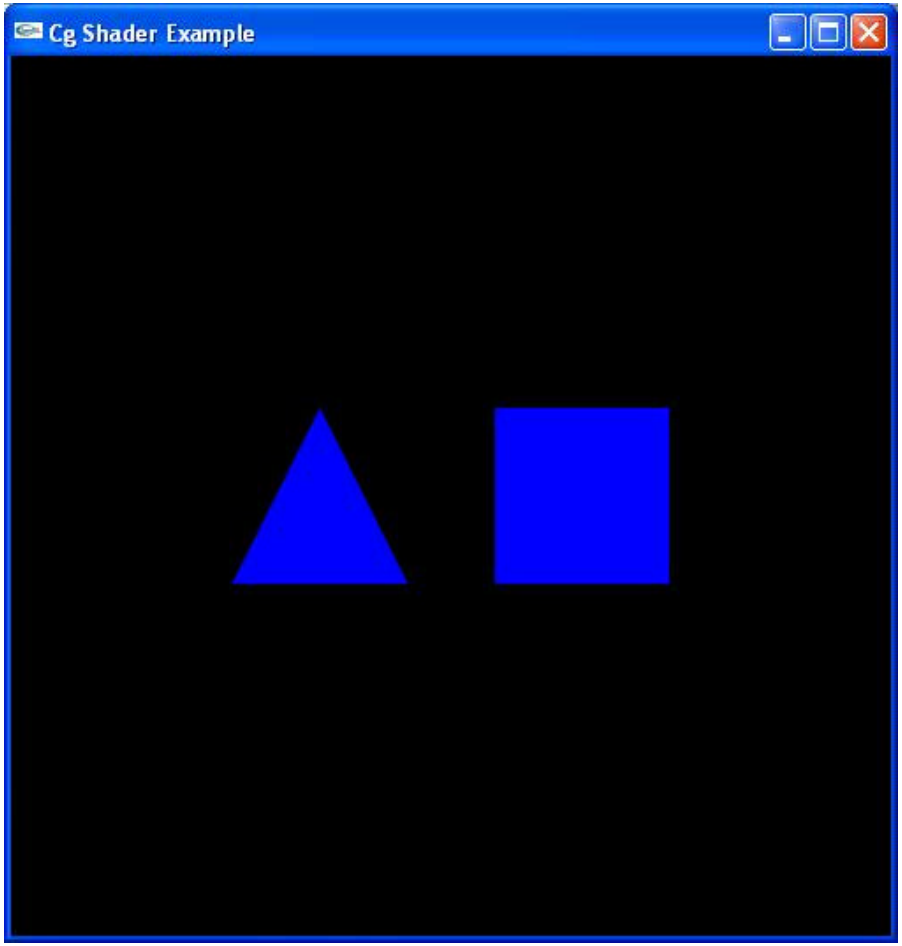
Cg Keywords

dword*	new	static_cast	vertexshader*
dynamic_cast	operator	string*	virtual
else	out	struct	void
emit	packed	switch	volatile
enum	pass*	technique*	while

From “The Cg Tutorial” Appendix D

Sample Program

- Let’s discuss the sample program.

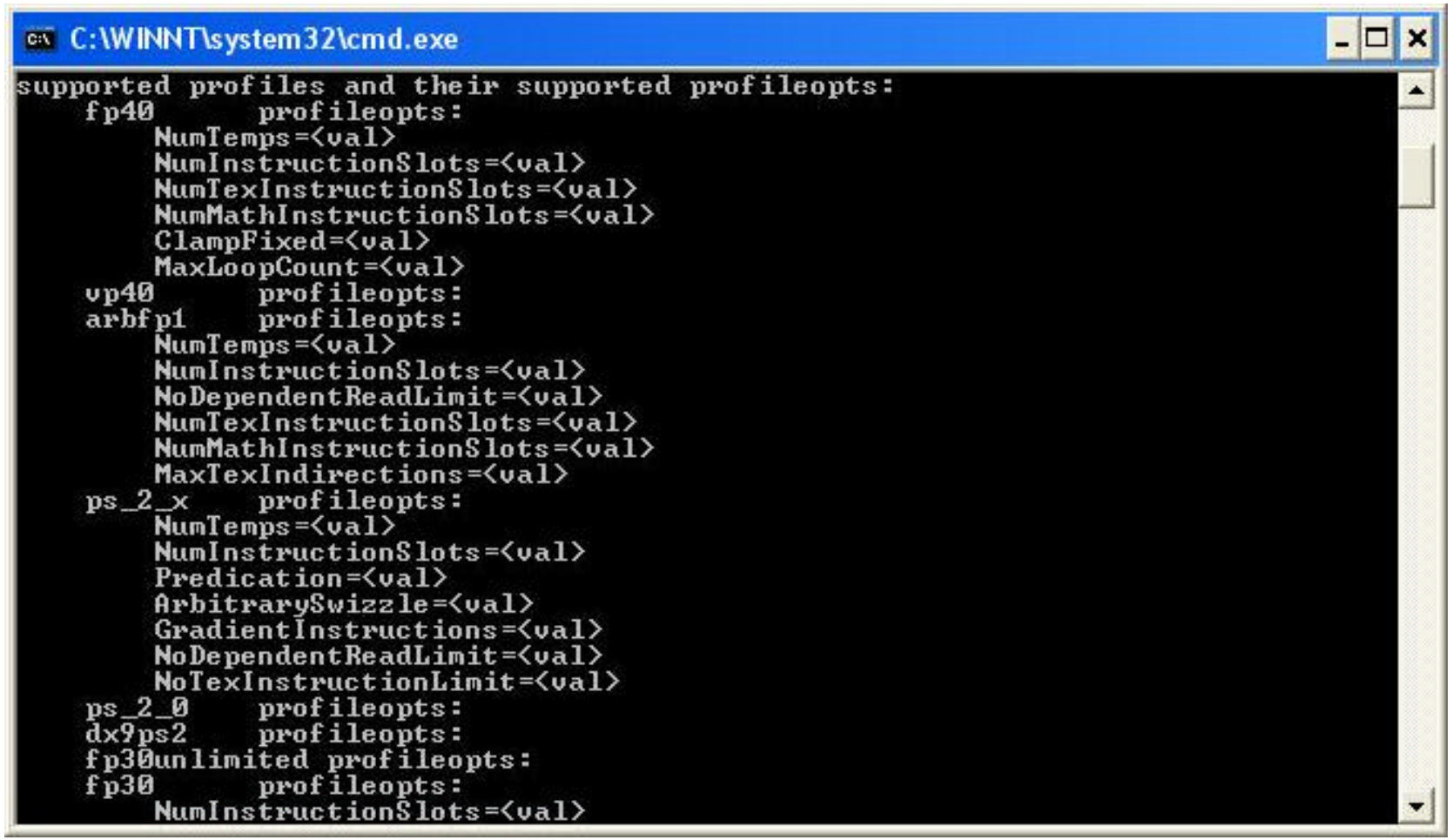


What we need…

- Compiler with a good IDE.
- The Cg Compiler, RunTime, and supporting libraries.
- Latest drivers for your GPU.
- Typing “cgc –help” at the command prompt will display a list of available profiles in the compiler.
- glGetString(GL_EXTENSIONS) will return extensions supported by the hardware.

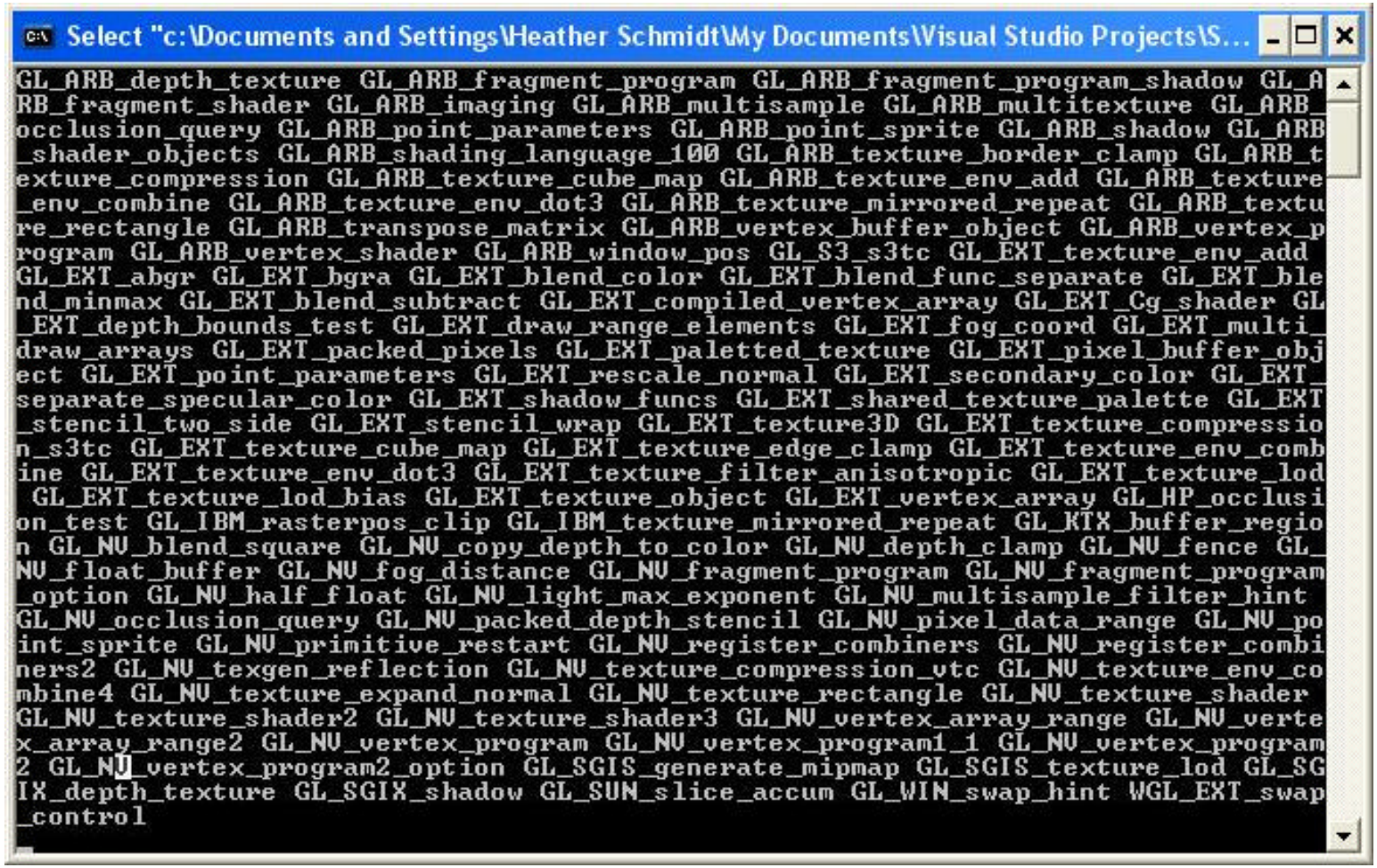
CGC Profiles

- Supported Profiles



Extensions

- Extensions

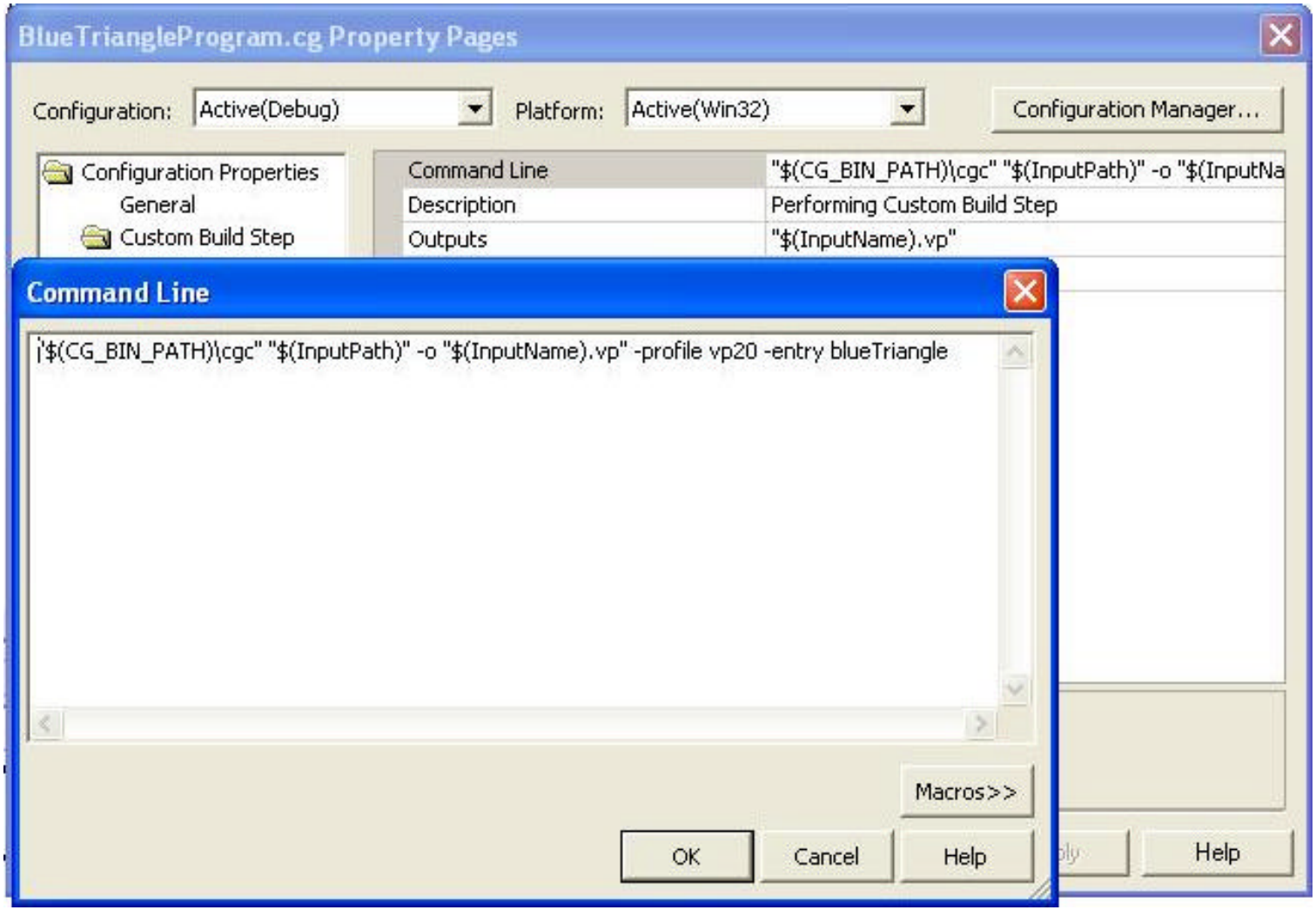


Compiling a Shader

- Shaders can be compiled statically or dynamically.
- Static compilation may save time at initialization.
- Dynamic compilation allows better profiles and future optimizations.
- A good practice is to statically compile a shader to remove errors; then, dynamically compile with application.

Configuring .NET

- This quick fix will work for now.



Textures

- Vertex Programs use TEXCOORD semantics to output one or more texture coordinate sets.
- Fragment Programs then use the texture coordinates.
- Cg provides “sampler” types.
- The tex*() function allows samplers and texture coordinates to be used to access textures. E.g. *tex2D(decal, texCoord);*

Let There be Light

- The standard fixed function pipeline implements per vertex light.
- A fragment program can implement per pixel lighting.
- Spotlights, distance attenuation, and directional lights are other possible effects.

Another Vertex Program

```
void simpleTransform(float4 objectPosition : POSITION,
                     float4 color          : COLOR,
                     float4 decalCoord     : TEXCOORD0,
                     float4 lightMapCoord  : TEXCOORD1,
                 out float4 clipPosition   : POSITION,
                 out float4 oColor         : COLOR,
                 out float4 oDecalCoord    : TEXCOORD0,
                 out float4 oLightMapCoord : TEXCOORD1,
             uniform float brightness,
             uniform float4x4 modelViewProjection)
{
  clipPosition = mul(modelViewProjection, objectPosition);
  oColor = brightness * color;
  oDecalCoord = decalCoord;
  oLightMapCoord = lightMapCoord;
}
```

From “Cg in 2 Pages”

(page 1 in case you can’t find it.)

A Fragment Program

```
float4 brightLightMapDecal(float4 color          : COLOR,
                           float4 decalCoord     : TEXCOORD0,
                           float4 lightMapCoord  : TEXCOORD1,
                     uniform sampler2D decal,
                     uniform sampler2D lightMap) : COLOR
{
  float4 d = tex2Dproj(decal, decalCoord);
  float4 lm = tex2Dproj(lightMap, lightMapCoord);
  return 2.0 * color * d * lm;
}
```

From “Cg in 2 Pages”

(page 1 in case you can’t find it.)

More Advanced Topics

- Animation
- Particle Systems
- Environment Mapping
- Bump Mapping
- Fog
- Projective Textures
- Shadows
- General Purpose Computation

Cg Demos

- In case I still have a lot of time left, here's a cool video!

Other Tools

- http://developer.nvidia.com/object/fx_composer_home.html
- http://developer.nvidia.com/object/nv_texture_tools.html
- http://developer.nvidia.com/object/nvperfhud_home.html
- http://developer.nvidia.com/object/nvshaderperf_home.html
- http://developer.nvidia.com/object/melody_home.html

References

- T. Akenine-Möller, E. Haines, “Real-Time Rendering, 2nd ed., A K Peters, 2002
- R. Fernando, M. Kilgard, “The CG Tutorial”, Addison-Wesley Professional, 2003
- http://developer.nvidia.com/
- M. Kilgard, “Cg in Two Pages”, 2003