

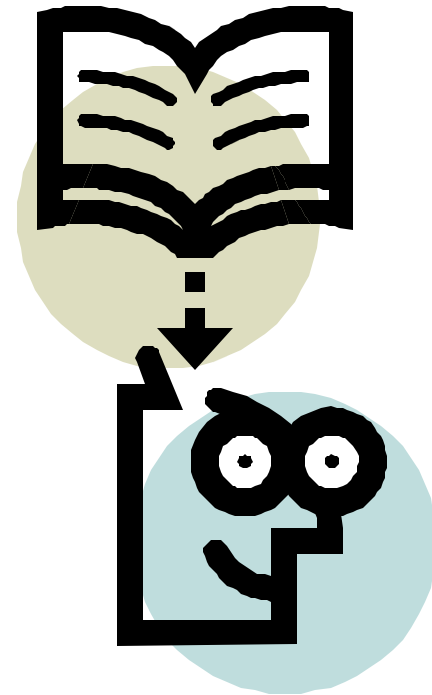


**CS 563 Advanced Topics in  
Computer Graphics**  
*Shader Programming with Cg*

by Michael Schmidt

# Outline

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





## History of Shaders

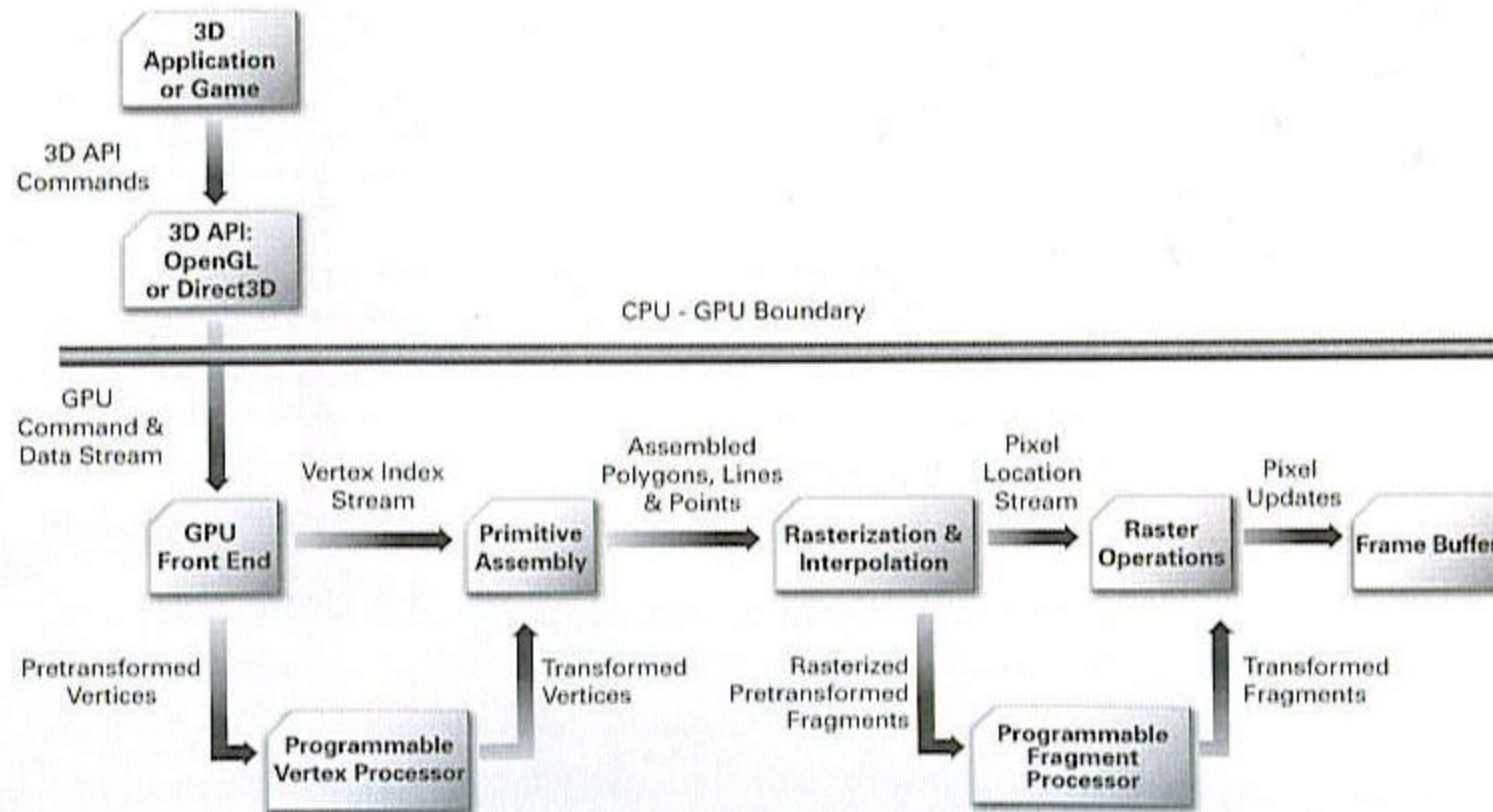
- RenderMan - offline.
- PixelFlow and RealTime Shading Language – real-time.
- Current languages are their descendants.
- programmable GPUs = real-time shaders.
- Shaders can be written in assembly, but there are huge benefits to using a high level language.
- Four classes of operations: constant for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

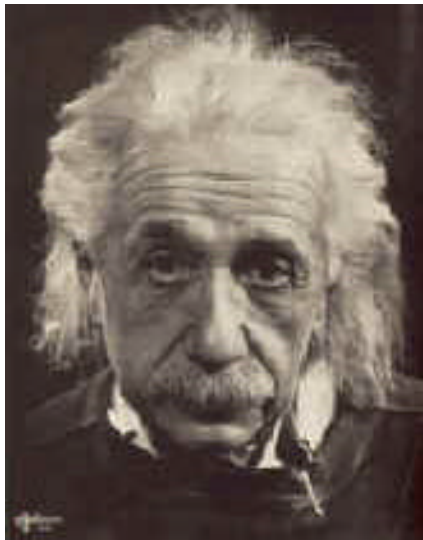
- Cg supports new special “packed” types such as float4 and float4x4.
- The GPU performs operations on packed types efficiently.
- Prefer packed types over arrays.
- Semantics tell the graphics hardware how an identifier should be used.
- Functions can be entry functions or internal functions.
- “Uniform”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

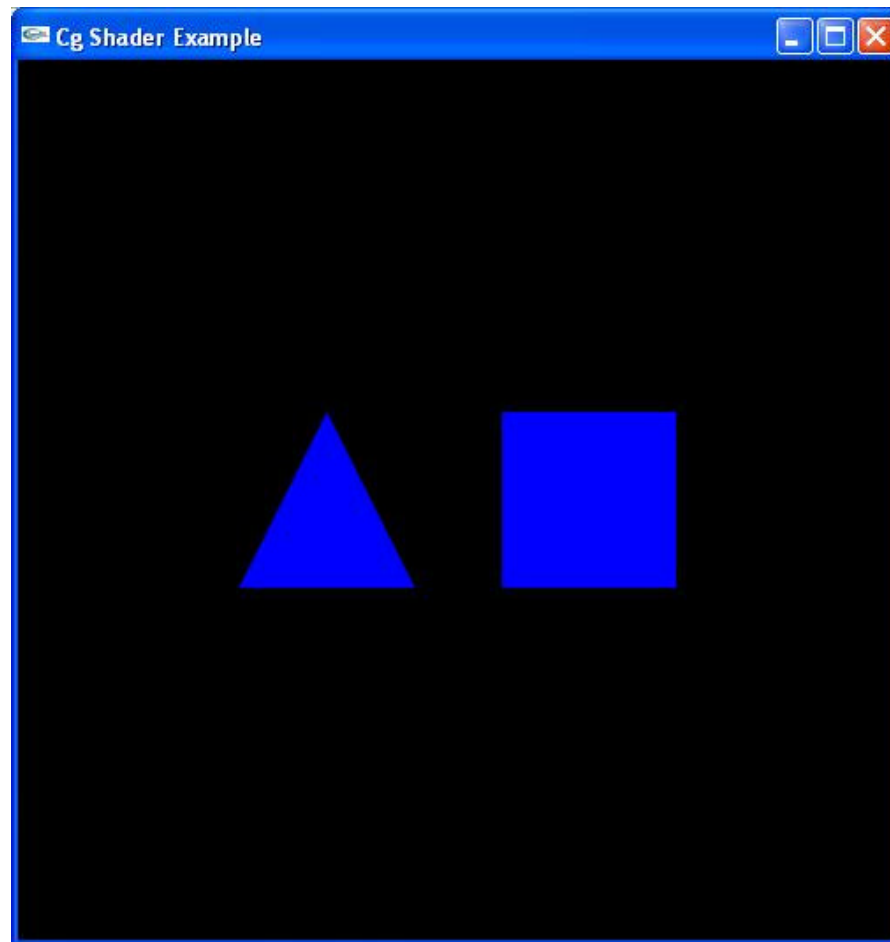
# Cg Keywords

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.



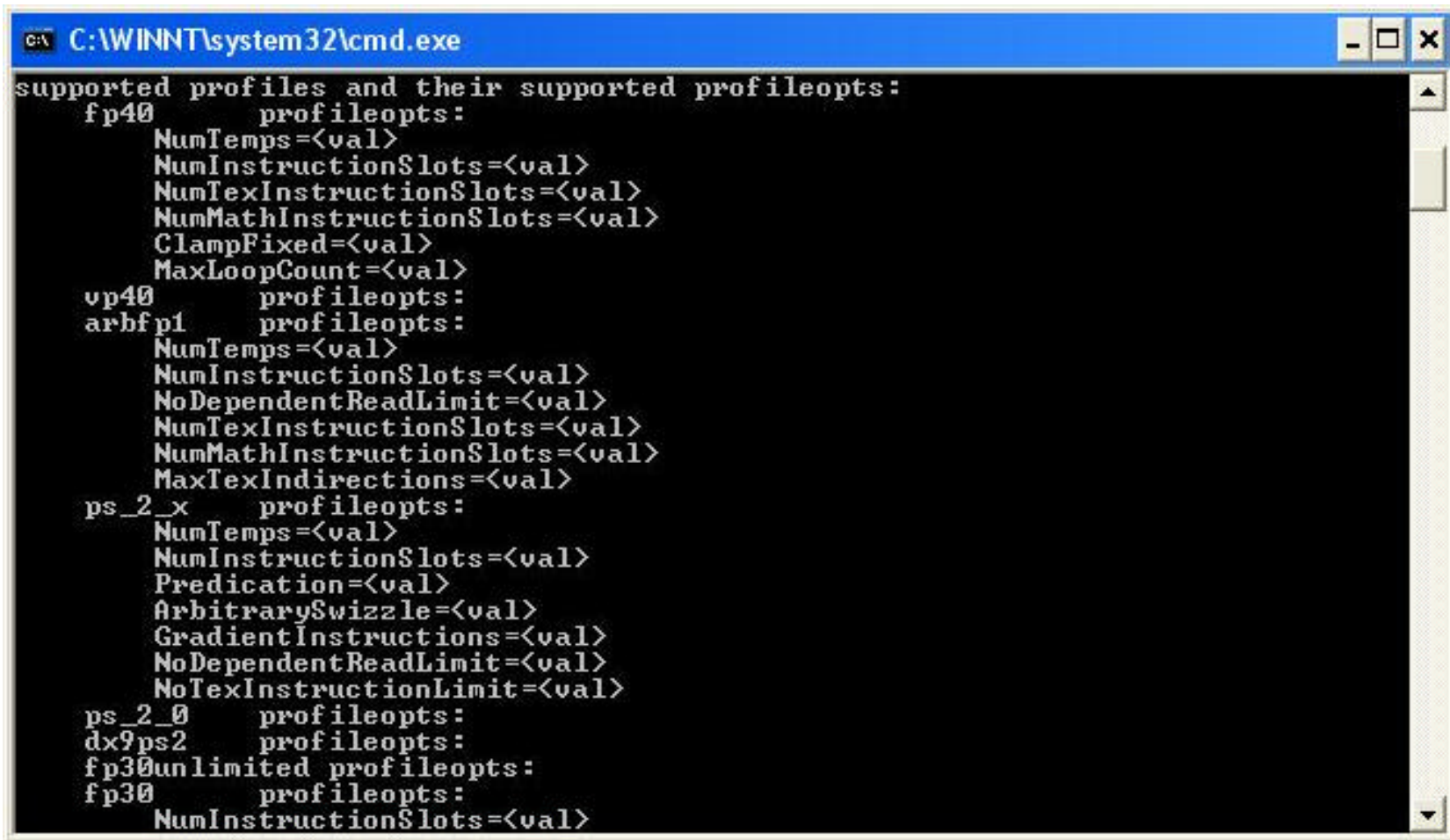


## What we need...

- 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

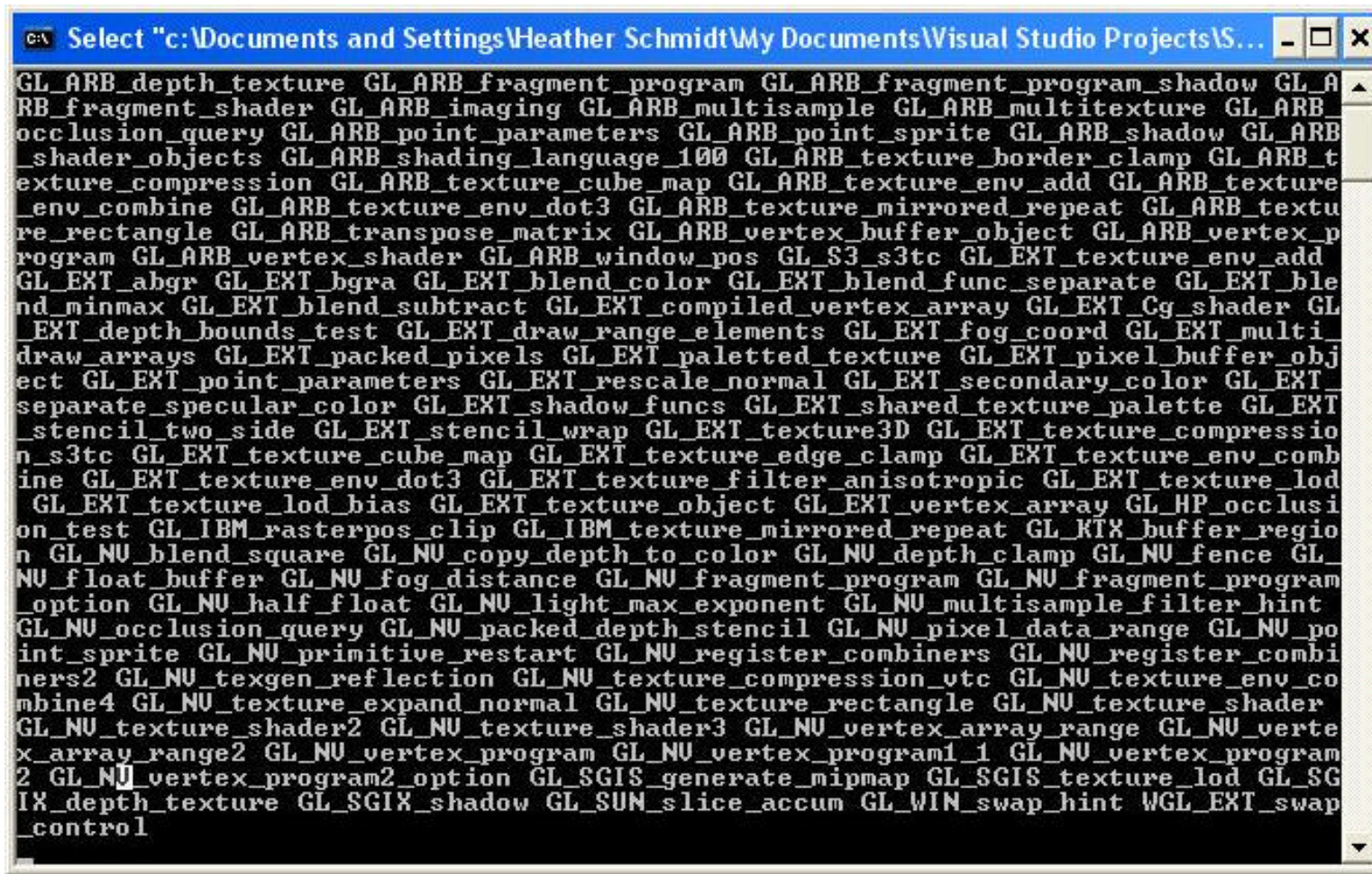
- Supported Profiles



```
C:\WINNT\system32\cmd.exe
supported profiles and their supported profileopts:
fp40      profileopts:
  NumTemps=<val>
  NumInstructionSlots=<val>
  NumTexInstructionSlots=<val>
  NumMathInstructionSlots=<val>
  ClampFixed=<val>
  MaxLoopCount=<val>
vp40      profileopts:
arbf p1    profileopts:
  NumTemps=<val>
  NumInstructionSlots=<val>
  NoDependentReadLimit=<val>
  NumTexInstructionSlots=<val>
  NumMathInstructionSlots=<val>
  MaxTexIndirections=<val>
ps_2_x     profileopts:
  NumTemps=<val>
  NumInstructionSlots=<val>
  Predication=<val>
  ArbitrarySwizzle=<val>
  GradientInstructions=<val>
  NoDependentReadLimit=<val>
  NoTexInstructionLimit=<val>
ps_2_0     profileopts:
dx9ps2     profileopts:
fp30unlimited profileopts:
fp30      profileopts:
  NumInstructionSlots=<val>
```

# Extensions

- Extensions



```
GL_ARB_depth_texture GL_ARB_fragment_program GL_ARB_fragment_program_shadow GL_A
ARB_fragment_shader GL_ARB_imaging GL_ARB_multisample GL_ARB_multitexture GL_ARB
occlusion_query GL_ARB_point_parameters GL_ARB_point_sprite GL_ARB_shadow GL_ARB
_shader_objects GL_ARB_shading_language_100 GL_ARB_texture_border_clamp GL_ARB_t
exture_compression GL_ARB_texture_cube_map GL_ARB_texture_env_add GL_ARB_texture
_env_combine GL_ARB_texture_env_dot3 GL_ARB_texture_mirrored_repeat GL_ARB_textu
re_rectangle GL_ARB_transpose_matrix GL_ARB_vertex_buffer_object GL_ARB_vertex_p
rogram GL_ARB_vertex_shader GL_ARB_window_pos GL_S3_s3tc GL_EXT_texture_env_add
GL_EXT_abgr GL_EXT_bgra GL_EXT_blend_color GL_EXT_blend_func_separate GL_EXT_ble
nd_minmax GL_EXT_blend_subtract GL_EXT_compiled_vertex_array GL_EXT_Cg_shader GL
_EXT_depth_bounds_test GL_EXT_draw_range_elements GL_EXT_fog_coord GL_EXT_multi
draw_arrays GL_EXT_packed_pixels GL_EXT_paletted_texture GL_EXT_pixel_buffer_obj
ect GL_EXT_point_parameters GL_EXT_rescale_normal GL_EXT_secondary_color GL_EXT_
separate_specular_color GL_EXT_shadow_funcs GL_EXT_shared_texture_palette GL_EXT
_stencil_two_side GL_EXT_stencil_wrap GL_EXT_texture3D GL_EXT_texture_compressio
n_s3tc GL_EXT_texture_cube_map GL_EXT_texture_edge_clamp GL_EXT_texture_env_comb
ine GL_EXT_texture_env_dot3 GL_EXT_texture_filter_anisotropic GL_EXT_texture_lod
GL_EXT_texture_lod_bias GL_EXT_texture_object GL_EXT_vertex_array GL_HP_occlusi
on_test GL_IBM_rasterpos_clip GL_IBM_texture_mirrored_repeat GL_KTX_buffer_regio
n GL_NV_blend_square GL_NV_copy_depth_to_color GL_NV_depth_clamp GL_NV_fence GL
NV_float_buffer GL_NV_fog_distance GL_NV_fragment_program GL_NV_fragment_program
_option GL_NV_half_float GL_NV_light_max_exponent GL_NV_multisample_filter_hint
GL_NV_occlusion_query GL_NV_packed_depth_stencil GL_NV_pixel_data_range GL_NV_po
int_sprite GL_NV_primitive_restart GL_NV_register_combiners GL_NV_register_combi
ners2 GL_NV_texgen_reflection GL_NV_texture_compression_vtc GL_NV_texture_env_co
mbine4 GL_NV_texture_expand_normal GL_NV_texture_rectangle GL_NV_texture_shader
GL_NV_texture_shader2 GL_NV_texture_shader3 GL_NV_vertex_array_range GL_NV_verte
x_array_range2 GL_NV_vertex_program GL_NV_vertex_program1_1 GL_NV_vertex_program
2 GL_NV_vertex_program2_option GL_SGIS_generate_mipmap GL_SGIS_texture_lod GL_SG
IX_depth_texture GL_SGIX_shadow GL_SUN_slice_accum GL_WIN_swap_hint WGL_EXT_swap
_control
```

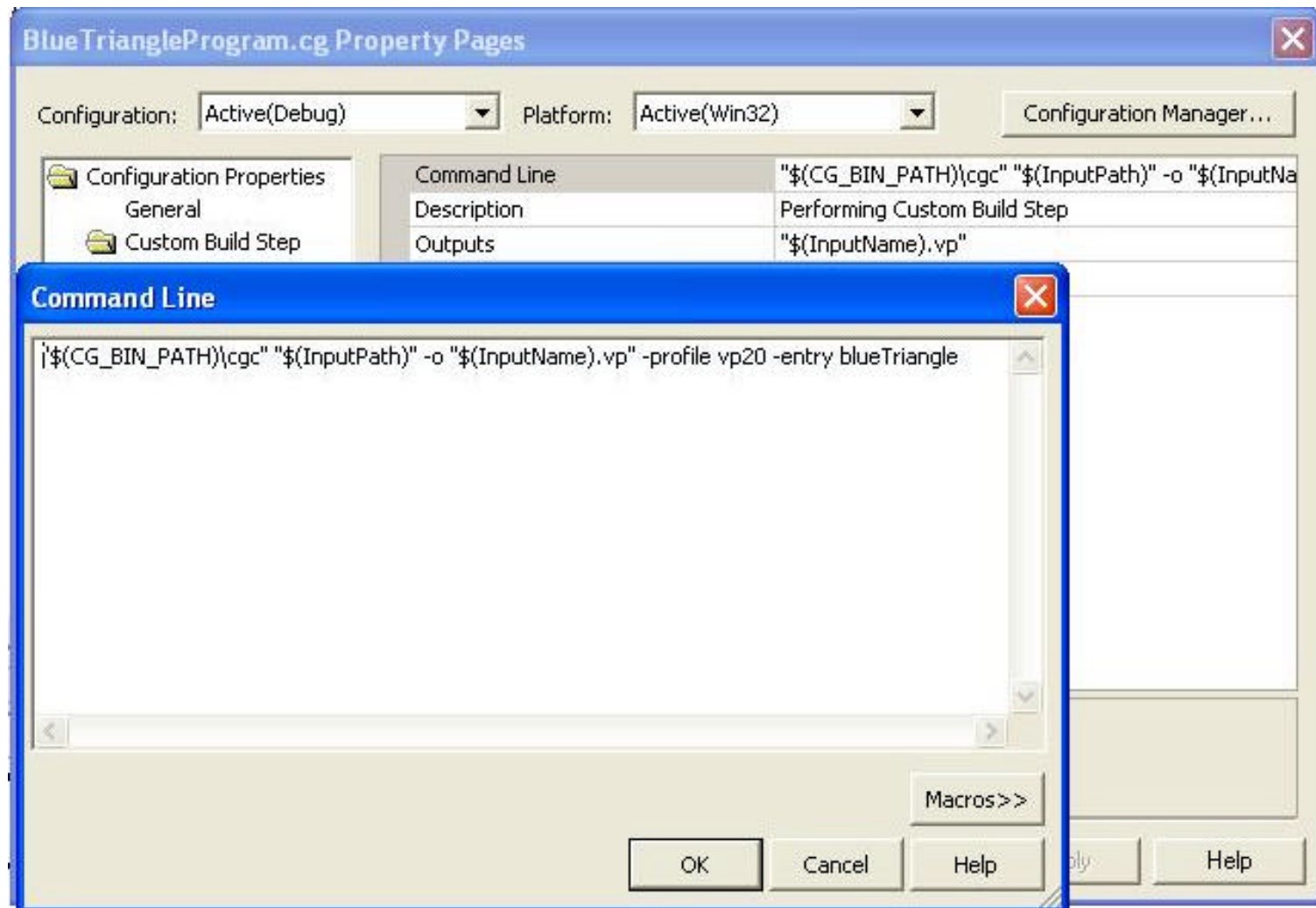


## Compiling a Shader

- 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.



- 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(decals, 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/fx_composer_home.html)
- [http://developer.nvidia.com/object/nv\\_texture\\_tools.html](http://developer.nvidia.com/object/nv_texture_tools.html)
- [http://developer.nvidia.com/object/nvperfhud\\_home.html](http://developer.nvidia.com/object/nvperfhud_home.html)
- [http://developer.nvidia.com/object/nvshaderperf\\_home.html](http://developer.nvidia.com/object/nvshaderperf_home.html)
- [http://developer.nvidia.com/object/melody\\_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