Dragonfly: A C++17 OpenGL Framework

Csaba Bálint csabix@inf.elte.hu Róbert Bán bundas@inf.elte.hu

Eötvös Loránd University, Budapest, Hungary



1. Introduction

Computer graphics developers and researchers usually have to choose between graphics APIs that are high performing or frameworks that are high level. In the former case, built-in debugging and developing tools are usually lacking. Current high level frameworks, however, are complicated to use or suffer performance overhead and sometimes are even harder to debug.

Dragonfly combines high performance and short client code while providing built-in debug tools. The framework achieves this via C++17 templates trading compilation time for the near-zero runtime overhead. Additionally, our framework can generate GUIs for most classes allowing runtime monitoring and debugging.

2. Overview

The abstraction encapsulates most OpenGL operations in five classes: Shader Program, Framebuffer, Texture, VertexArrayObject, and Buffer. A single line of Dragonfly code can represent a pipeline of rendering commands. A simplified post-process rendering code example is presented below.



		, , , , ,		
Texture		 	Buffers	
2D, 3D, Cube, ect.			Binary GPU storage	

4. Textures

In Dragonfly, all Texture classes are parameterized by the template parameter InternalFormat. This allows for both compile and runtime optimizations and additional debugging.

auto myTextureFromFile = df::Texture2D<df::u16vec3>("Lena.jpg"); // 16 bit per channel RGB image. Default is 8 bit. auto myCubeMap = df::TextureCube<>("xpos.png", "xneg.png", "ypos.png", "yneg.png", "zpos.png", "zneg.png"); auto myTextureView = myCubeMap[df::X_POS]; // 2D texture view of a single face with zero GPU memory cost.

Apart from the usual texture operations, OpenGL texture views are exploited to allow memory efficient and easy reuse of textures. Texture classes safely allocate immutable GPU storage reducing stress due to driver overhead.

5. Shader Programs

Shader programs are compiled from multiple shader files binned into shader stages enabling large shader code libraries. Even though the shader stages are compile time information that are used for optimization and static assertion in Dragonfly, the shader code can be changed and recompiled at

3. Previous work

Amongst OpenGL libraries and frameworks, there are scene graphs: GizmoSDK, Open Scene Graph (Vulkan Scene Graph), OpenSG, Open Inventor; and there are graphics engines such as Irrlicht, Ogre3D, Visualization Library, and even game engines: ClanLib, Delta3D, and Panda3D. However, none of these are suited for research purposes because development is slow or the library is not flexible enough for developing algorithms for computer graphics. NVIDIA Falcor [1] is a mature graphics C++ framework for research that supports Vulcan and DirectX 12 with the DXR Raytracing API.

runtime.

auto myMeshProgram = df::Program("meshprog.vert"_vert + "shadingHeader.glsl"_frag + "meshprog.frag"_frag); auto myPostProcess = df::ProgramEditor("postprocess.vert"_vert + "postprocess.frag"_frag); //Generates GUI

The ProgramEditor allows runtime shader code editing and compilation. Uniform variables sent to the GPU may be observed or even overwritten to help with shader code debugging.

▼ sdf-trace-prog	▼ so	df-tra	ce-prog						
				Compile & Link Fragment editor Vertex editor Uniform editor Subroutine editor					
▼ File Editor [Shaders/types.gls]]		0	int	debugMode	0				
Save Load		1	int	gAOMaxIters	5				
1 #define RAY(r,t) (r.Direction*(t)+r.Origin)		2	float	gAOStepSize	0.400000				
<pre>2 #define SDF(r,t) sdf(RAY(r,t)) 3 #define CONE_SDF(raycone, t) ((SDF(raycone.ray,t)-(t)*raycone.tana-raycone.rad)/(1.0+raycone.tana)) 4 #define CONE_SDF_FOS(fos, raycone, t) (((fos)-(t)*raycone.tana-raycone.rad)/(1.0+raycone.tana))</pre>		3	float	gAOStrength	0.70000				
<pre>4 #define conc_sor_ros(ros, ragcone, c) (((ros)-(c)*ragcone.cana-ragcone.rau))(1.0+ragcone.cana)) 5 6 struct RayDesc</pre>		4	vec3	gAmbient	=(0.010000,0.010000,0.011000)				
7 { 8 vec3 Origin;		5	vec3	gCameraDir	=(0.763298,-0.406164,0.502402)				
9 float Tmin; L0 vec3 Direction;		6	vec3	gCameraPos	=(-1.000000,0.500000,-0.800000)				
.1 float Tmax ´ .2 };		7	vec3 st	ruct glm::vec<3,float,0>	=(1.400000,1.410000,1.420000)				
3 4 struct RayCone Error at line 12: 5 syntax error, unexpected '}', expecting ',' or ';' at token "}"		8	vec3	gCookRoughness	=(0.030000,0.034000,0.032000)				
5 {		9	vec2	gDepthcalcCoeffs	=(1.000100,-0.100010)				
7 float tana; //Cone opening angle when cone tracing can be zero 8 float rad; //Cone starting radius at Origin (t=0)		10	vec3	gDiffuse	=(0.680000,0.780000,0.920000)				
9 }; 20 21 struct TraceResult 22 { 23 float T; // Distance taken on ray 24 int flags; // bit 0: distance condition: true if travelled to far t > t_max		11	mat4	gInverseViewProj	-0.321049 0.111439 4.999502 -4.237204 0.000000 0.300155 -2.499751 2.094086 0.487769 0.073349 3.999602 -3.497999 0.000000 -0.000000 -4.999502 5.000502				
25 // bit 1: surface condition: true if distance to surface is small < error thre: 26 // bit 2: iteration condition: true if took too many iterations		12	vec3	gLightPos	=(2.000000,7.000000,10.000000)				
27 // int iterations; // Iterations taken 28 // float distance; // Distance to the surface (lower bound)		13	vec2	gNearFarClips	=(0.100000,1000.000000)				
9 }; :0		14	vec3	gNormEps	=(0.032258,0.032258,0.032258)				
11 struct SphereTraceDesc 12 { 		15	float	minStepDist	0.032258				
33 float epsilon; //Cone stopping distance to surface 24 ist maxitance //Maximum itenation count		16	int	refineRoot	1				
File Editor [Shaders/Primitives/common.gls1]		17	float	sDistMul	1.032258				
• File Editor [Shaders/SDF/sdfTexture3D0thOrder.gls1]		18	float	sOffset	0.000000				
► File Editor [Shaders/SDF/weightBlend.gls]]		19	float	sScale	1.00000				
► File Editor [Shaders/sdf.gls1]									

7. Conclusions

	Dragonfly	Falcor $[1]$	
Overhead	Very low	Low	
Functionality	Enough	A lot	
Reliability	TBD	Battle tested	
Debugablitiy	High	Low	
Code brevity	Very short	Short	

Dragonfly is best suited for research because one can quickly develop computer graphics algorithms and also benchmark them. However, Dragonfly does not support hardware accelerated raytracing and applications that benefit from heavily multi-threaded graphics API usage due to the design of OpenGL.

Therefore, we claim that code brevity, high performance, and the ease of debugging graphics algorithms outweight the drawbacks of lack of functionality. Future work include full OpenGL support, GLSL code optimizations and practical debug tools using the compile time information.

6. Framebuffers

A FramebufferObject class may be constructed from multiple Texture2D and Renderbuffer objects. Only complete framebuffers can be targets of a rendering pipeline, thus the class contains all the compile time information about its output attachments. Henceforth, Dragonfly can decide which OpenGL commands are needed during compilation rather than runtime.

The two examples above demonstrate the usage of frambuffers for postprocess and deferred shading. A compile time warning and a runtime assertion in debug configuration are issued because the GPU implementation may not support 16 bit floating format as a framebuffer output.

8. References

[1] Nir Benty, Kai-Hwa Yao, Lucy Chen, Tim Foley, Matthew Oakes, Conor Lavelle, and Chris Wyman. The Falcor rendering framework, 10 2019. https://github.com/NVIDIAGameWorks/Falcor.

Acknowledgement Supported by the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology.

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00001).