JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

A SURVEY ON OPENGL

[1]Vennila M , [2]Jenifer I

[1]Lecturer, [2]Assistant Professor

[1]Game Programming Department, [2]Computer Science Department

[1]ICAT Design and Media College, Chennai, [2]Parvathy's Arts and Science College, Dindigul

1. INTRODUCTION

The primary option for rendering visuals in many 3D games is OpenGL. The dedicated GPU (Graphics Processing Unit) on modern computers has its own memory to speed up the rendering of graphics. Application Programming Interface for Graphics, Cross-Platform. There is no input, audio, orwindowing API provided by OpenGL; it just focuses on rendering.

Keywords: Graphics API, Rendering, OpenGL.

2. CORE-PROFILE vs IMMEDIATE MODE

In the old days, using OpenGL meant developing in immediate mode (or fixed function pipeline) which was an easy-to-use method for drawing graphics. Most of the functionality of OpenGL was hidden inside the library and developers did not have much control over how it did its calculations. For that reason, the specification started to deprecate immediate mode functionality from version 3.2 onwards.

When using the modern version of the graphics programming language, learning how to use it requires a much better understanding of graphics programming. The advantage of learning the modern approach is that it is very flexible and efficient but it's also more difficult to learn. The immediate mode abstracted quite a lot from the actual operations OpenGL performed and while it was easy to learn, it was hard to grasp how it actually operates.

Five sets of libraries in our OpenGL programs:

A. **Core OpenGL (GL):** consists of hundreds of commands that start with "gl" (e.g., glColor, glVertex, glTranslate, glRotate). Using a collection of geometric primitives including point, line, and polygon, Core OpenGL represents an object.

B. **OpenGL Utility Library (GLU):** developed on top of the fundamentals of OpenGL to offer necessary utilities (such adjusting camera view and projection) and more model-building (such as quadric surfaces and polygon tessellation). Commands using GLU begin with the letter "glu" (e.g., gluLookAt, gluPerspective).

C. **OpenGL Utilities Toolkit (GLUT):** The operating system and windowing system are not intended to affect OpenGL in any way. In addition to providing more building models, GLUT is required to interact with the Operating System (such as opening windows and managing key and mouse inputs) (such as sphere and torus). GLUT commands begin with the "glut" prefix (e.g., glutCreatewindow, glutMouseFunc). Platform-specific OpenGL extensions like GLX for the X Window System, WGL for Microsoft Windows, and AGL, CGL, or Cocoa for Mac OS are the foundation for GLUT, a platform-independent OpenGL extension.

D. **OpenGL Extension Wrangler Library (GLEW):**"Open-source C/C++ extension loading library GLEW is cross-platform. In order to ascertain which OpenGL extensions are supported on the target platform, GLEW offers effective run-time techniques."

E. **GLFW (Graphics Libary Framework):** is a C library that was created with OpenGL in mind. The minimum necessities needed to render goods on the screen are provided by GLFW. For our purposes, it is sufficient to allow us to establish window parameters, build an OpenGL context, and handle user input.

3. GLSL:

The primary shading language for OpenGL is the OpenGL Shading Language (GLSL). While numerous shading languages can be used with OpenGL supported by OpenGL Extensions, GLSL (and SPIR-V) are directly supported by OpenGL without extensions. GLSL, a language comparable to C, is used to create shaders. GLSL has helpful capabilities geared primarily at manipulating vectors and matrices that are designed for use with graphics.

GLSL also features two container types, namely vectors and matrices.

a) Vectors

In GLSL, a vector is a 2, 3, or 4-component container for any of the previously listed fundamental kinds. They may appear as follows:

- vecn: the default vector of n floats.
- bvecn: a vector of n booleans.
- ivecn: a vector of n integers.
- uvecn: a vector of n unsigned integers.
- dvecn: a vector of n double components.

b) Matrix

A typical 4x4 matrix used as a transformation matrix for 3D homogeneous coordinates is known as an OpenGLMatrix. The OpenGL high-performance graphics standard primarily utilizes the data architecture of an OpenGLMatrix.

- MatrixF emptyMatrix(int numRows, int numCols)

4. SHADERS

Small applications that run on the GPU are called shaders. Each individual segment of the graphics pipeline is run through one of these applications. Shaders are essentially just programs that convert inputs into outputs. Because they are not permitted to communicate with one another and can only do so through their inputs and outputs, shaders are also quite solitary programs.

Specific elements of the rendering pipeline are defined as being programmable. Each of these phases or parts denotes a specific programmable processing method. There are specific inputs and outputs for every stage that are transmitted from earlier stages to the subsequent stage.

OpenGL Shading Language is the language used to create shaders. The following shader stages, along with their enumerator names, are defined by the OpenGL rendering pipeline:

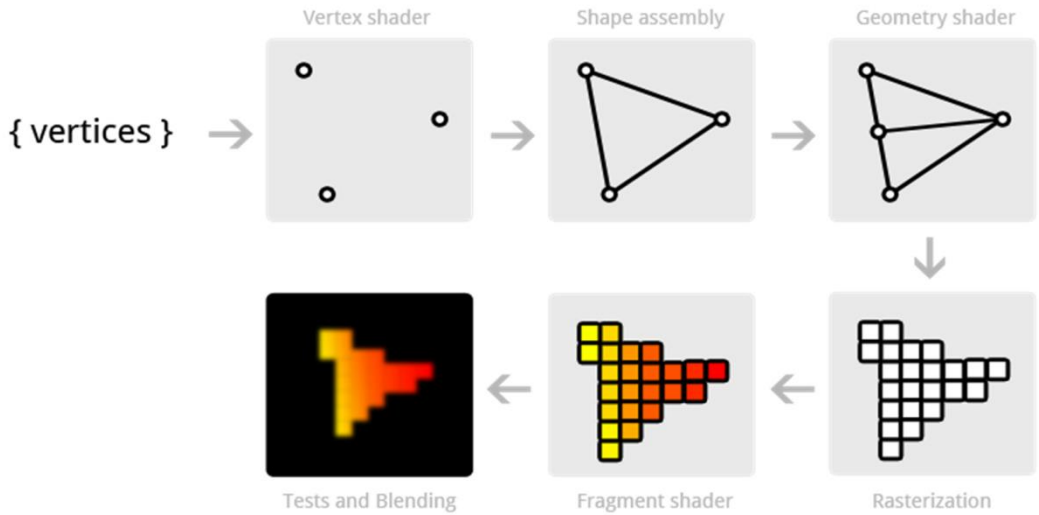
- **Shaders for Vertex GL_VERTEX_SHADER**
- **Shaders for Tessellation Control and Evaluation GL_TESS_CONTROL_SHADER and GL_TESS_EVALUATION_SHADER.**
- **Shaders for geometry: GL GEOMETRY SHADER**
- **Shaders for fragments: GL FRAGMENT SHADER**
- **Shaders that compute: GL COMPUTE SHADER**

The currently bound Program Object, or Program Pipeline Object, will be used in the rendering process when a drawing command is executed. The shader code saved in the active program will be runby the programmable components of the rendering pipeline (s).

a) Vertex Shaders:

In the vertex stream, roughly once every input vertex. Due to the Post Transform Cache, this might happen less frequently per vertex if indexed rendering is employed, but it will happen atleast once for each distinct combination of vertex attributes.

Input: position of points in 3D spaceOutput: 2D projection of the



points

b) Tessellation Control Shaders

Each output vertex is exactly once for each patch. Through their output variables, invocations using the same input patch can communicate with one another.

c) Tessellation Evaluation Shaders

In the tessellation of the abstract patch, around once per vertex. In the patch, a certain vertex could be handled more than once. The maximum number of TES invocations is one for each vertex for each primitive generated by a patch, while the minimum number is one for each unique vertex in the patch.

d) Geometry Shaders

When a primitive reaches this level, once. For each input primitive, the GS might be called more many once thanks to geometry shader instancing.

e) Fragment Shaders

Once for each Fragment the rasterizer produces. As the implementation might utilize "helper" fragment shader objects, it might be run more frequently than this. However, these instances cannot write data (in any way, whether fragment shader outputs, Image Load Store or anything else). They are mostly used to compute implicit derivatives, which are necessary for many texture sampling functions.

f) Compute Shaders

The quantity of invocations is determined by multiplying the quantity of work groups requested by the dispatch operation by the local size of the compute shader. A work group's intercommunication capabilities for compute shader invocations are somewhat limited.

5. CONCLUSION

A vast range of resources is available to shaders. These have access to image variables, atomic counters, shader storage buffers, uniforms, uniform blocks, textures, and maybe other data as well. However, the amount of resources that each shader step can access is constrained. Each resource has a maximum number of available resources for each stage that can be queried.

Shaders are used in video games to precisely specify shadows, lighting, texture gradients, and other effects. But they are much more versatile than merely what their name suggests.

6. REFERENCES

1. Ms. Warsha M.Choudhari, Ms. Rinku Rajankar “A Survey Paper on OpenGL ES” International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169Volume: 3 Issue: 4 1755 - 1758

2. Alun, Evans, Marco, Romeo “3D graphics on the web: A survey” Computers & Graphics Volume 41, June 2014, Pages 43-61

3. By Tom McReynolds, David Blythe “Advanced Graphics Programming Using OpenGL”

4. Joey De Vries “Learn OpenGL Graphics Programming” June 2020 -ISBN: 978-90-90-33256-7

5. V. Scott Gordon, PhD, John L. Clevenger, “Computer Graphics Programming in OpenGL with C++” Stylus Publishing, LLC, 29-Nov-2018