

Question 2

Write a critical appraisal of OpenGL or, at least, those parts of OpenGL that we have covered in the course. Identify the strengths and weaknesses of OpenGL as you see them. You can give examples of source code (or even complete programs) that demonstrate good or bad features of OpenGL if you like. If you know another graphics API, such as Glide or Direct3D, you may make comparisons between them and OpenGL.

Since we have evolved to a technological age where hardware performance is not quite an issue as it was before, an emphasis has been put to provide high end output and graphics to the user. Thus, cards manufacturers and graphics developers have started working hand in hand to give out a good graphics standard for everyone to use. All began in the 1980s where Silicon Graphics used to manufacture very powerful graphics machines along with drivers to support them. These machines are no more considered powerful since what used to be high end then is now worse than consumer products. In 1992, OpenGL was created from the IRIS GL API for UNIX. OpenGL was a library to set open standards. It was written in C code to allow both C and C++ implementations. Later, Microsoft adopted OpenGL 1.2 and included it in windows. What are the goods and bad that OpenGL includes, and in what ways is it different from currently available graphics API's like D3D and GLIDE? Most of these will be discussed in this short paper.

OpenGL was engineered to satisfy graphics needs for the future. This means that most of the conceptions put in it back when it was written weren't available on graphics hardware to make use of it. As hardware improved, OpenGL didn't change dramatically to fit the needs of the new technologies. For example, consider T&L architecture, which is currently embedded in most graphics GPUs, wasn't there 3 years ago. However, OpenGL had the functions and methods to emulate translation

and lighting effects even without the hardware. This is called software rendering. OpenGL hides the hardware layer making it easy to program for all kinds of hardware and making sure that the same effect will show on all. This is defined by conformance. Moreover, considering different operating systems, OpenGL would do just fine on any of them (Portability) since it is open source. OpenGL has an architecture review board (ARB) which decides on the open standards. Although many big companies are involved in the ARB, they tend to set standards not to take advantage of them personally. One bad thing is that whoever holds a license to update OpenGL can publish it and then there may be many versions out there; even versions which are made to specific hardware cards. This makes it a bit non-standard. OpenGL has evolved through the years, since 1992, and now the most current usable version is OpenGL 1.4. OpenGL 1.5 specifications have come out but there are still corrections being made to it and in the near future, OpenGL 2.0 is being developed by 3DLabs. OpenGL 2.0 is thought to be another version that should set the standards for many years to come. As far as it is known, OpenGL 2.0 is a Shading language.

On the other hand, DirectX's D3D is considered as the competitor to OpenGL. D3D is developed and applied by Microsoft. Microsoft only has the privilege to update and release D3D versions. Moreover, what is being done is that there is a new version of D3D in at most every new year. This makes DirectX up-to-date with current technology, but at the cost of programmers needing to know the language updates as soon as they come out in order to use the newly available features in their software. Microsoft has been working with card manufacturers side by side to keep up with the coming technologies and include them in their new versions of DirectX. All the above concepts smash Microsoft DirectX's D3D in favor of OpenGL. We can say that OpenGL makes very large steps once in a while, while DirectX

changes very frequently and what programmers do not want is to keep on learning the new language rather than spending time on creating new ideas for their games and applications. This has also some good points if ever OpenGL is not up-to-date and needs a new set of standards to make up with time, which is the case nowadays and this is why 3DLabs is working on OpenGL 2.0. For more info about this matter check the following website:

<http://www.3dlabs.com/support/developer/ogl2/index.htm>.

Since most games are made on Windows OS platforms, then the use of DirectX is a common practice in the game industry. Moreover, DirectX works only in C++ because of its class and COM objects based architecture. This is different from OpenGL where everything is done in C (could be used in C++) and everything is procedural by calling some functions to do the job after passing it the needed attributes. In DirectX, working with COM technology has its goods and bads which is not in the topic of this paper. But one thing is that everybody likes programming in OpenGL because of its minimal use of code to do something. D3D started with 800 lines of code for initialization only with D3D version 5, Microsoft somehow corrected the mistake and code initialization was dropped down to 200 with D3D 7. However compared to OpenGL, this is way too much. One page of OpenGL code can make u up and running with no errors because of conformance on all hardware cards. D3D has to divide the classes of cards into cases and deal with each case separately. In the following example, OpenGL code is shown to create a triangle from 3 vertices. Same thing in D3D with is significantly a lot more code. D3D uses something called execute buffers. D3D programmers have to initialize everything and store the data and commands in arrays and then they pass the whole thing in a single call. This may seem easy and short, but it creates a lot of code and everything has to be defined before it is processed.

| OpenGL | D3D |
|--|---|
| <pre>glBegin (GL_TRIANGLES); glVertex (0,0,0); glVertex (1,1,0); glVertex (2,0,0); glEnd ();</pre> | <pre>(psuedo code, and incomplete) v = &buffer.vertexes[0]; v->x = 0; v->y = 0; v->z = 0; v++; v->x = 1; v->y = 1; v->z = 0; v++; v->x = 2; v->y = 0; v->z = 0; c = &buffer.commands; c->operation = DRAW_TRIANGLE; c->vertexes[0] = 0; c->vertexes[1] = 1; c->vertexes[2] = 2; IssueExecuteBuffer (buffer);</pre> |

Concerning which API is better and more performing, one can't really say. Both of them are top leading in the market and both work perfectly the same on a Windows OS machine and they cover the most important functionalities available in video cards today. However, as told before, OpenGL contains some features like the T-buffer (allows to calculate which texture to display first in terms of depth) which will be available very soon or are now in the newest and very expensive cards. And D3D has some features unavailable in OpenGL like specular blend at each vertex, color key transparency, and no clipping hints. The most important concepts like vertex arrays in OpenGL and vertex pools in D3D are basically equivalent. However, still not very used features like transparency and translucency are not near to get in D3D which makes OpenGL still ahead for some time now. DirectX doesn't run on non-Windows platform so basically to make sure your code works anywhere anytime, it's better to go with OpenGL. OpenGL, as said before, hides the hardware layer and provides a high level, clear, straightforward language which lets you express what you want to do. You don't have to know what hardware your program will be running on since conformance tests have been run on these cards by the manufacturers. It is a guarantee that whatever you develop will work on any compatible OpenGL card

and, moreover, will look the same on all. In D3D, once a program is done, it has to be tested on all the available cards on the market and some adjustments (actually a lot as described by one article) have to be made to the code so that it is up and working without problems. D3D is a layer between the hardware and the program tending more to be on the hardware programming side. D3D programmers have to consider different cases for different hardware and their features. Most programmers tend to develop with D3D because of Microsoft's market dominance. Anyways, everybody can choose his or her preference. It's a free world.

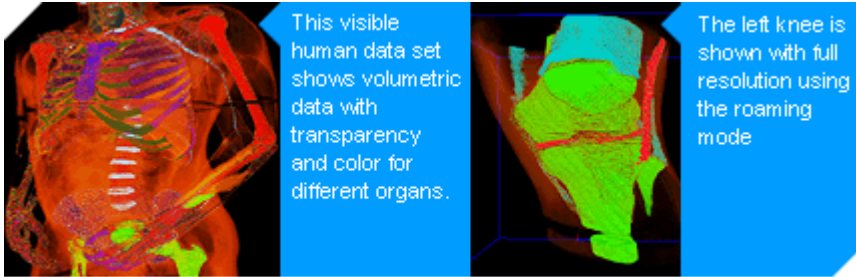
Both API are powerful and stable and there is no doubt that both of them are required to develop a good game on Windows PC, but only OpenGL would be used for anything else because of its high standards and precision. Some applications that use one and not the other are shown in the table below. And, it is known (as described by SGI itself) that all CAD, digital engineering and Military visualizations use OpenGL because of its reliability and platform independent structure.

| Direct3D | OpenGL |
|---|--|
| Simply 3D Unreal Tournament Earth 2150 Warcraft III Microsoft Flight Simulator Battle Isle 4 | ArchiCAD Autodesk (AutoCAD 2000) Quake (GL-Quake, Quake II & III) Microstation 95 MathGL3D (Mathematica) Maya (Character animation, Modeling) |

It is clear that OpenGL is used for more applications than games although it would do just fine with games. The fact is that DirectX contains a big set of tools to manipulate sound, input and network along with D3D. Games need a combination of all of these together to be interesting. Most of game programmers use Direct3D since they will eventually use the other features that come with it like DirectSound, DirectInput and DirectPlay. Applications like CAD tend to be purely graphical and OpenGL is highly accurate and easier to program with than D3D. There are plenty of topics and opinions on which is better and they go into the very details of the

languages. These can be found at the references section of this paper. One aspect is that by combining D3D and OpenGL, it would be better for all and there would be one and only one language to deal with. D3D retained mode is capable of sitting on top of OpenGL since it is a higher level language, unlike D3D Immediate mode which from it comes all the problems and cases. New kits are coming up for OpenGL and they are being developed by the graphics leading company SGI. These packages are described in the following table. Like DirectSound, OpenAL is an audio library to be part of the Open family. However, we do not hear about it a lot. I guess because nobody is really working on it seriously. OpenML is a library intended to contain all Open family components like DirectX contains Direct3D, DirectSound, etc... The bottom point is that it does not matter which API you program with. Both APIs are good and if you learn one, it would be easy to learn the other since the core concepts are the same.

As for GLIDE, they say it was a really good and straightforward library like OpenGL. But it only works on 3DFX chipsets that support it! Programmers tend to develop something for the whole Market rather than specific cards although 3DFX are good ones. But the market contains a lot of different cards and types so programming in GLIDE is not common. 3DFX chipsets are included in Creative Labs, TNT, and Voodoo Hardware which are not the best sellers today compared to NVidia, ATI, etc... There are rumors that 3DFX is fusing GLIDE into OpenGL to create a better library. NVidia has developed its own graphics library called Cg which is C-like and handles a lot of complex programming in easy straightforward functions. It is similar to OpenGL structure. Cg is most commonly being referred to as a shading language.

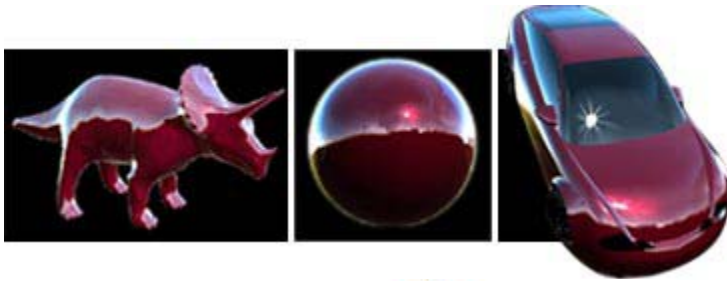


This visible human data set shows volumetric data with transparency and color for different organs.

The left knee is shown with full resolution using the roaming mode

The figures were generated using 3D clip textures supported by OpenGL Volumizer. The size of the visible male data set is 6.77 GB.

OpenGL Volumizer: used for treating volumetric the same way as you do for surface data.



OpenGL Shader: for appearance modeling and visual effects



OpenGL Optimizer: for digital modeling and prototyping

OpenGL Performer: for real-time visualizm

OpenGL Multipipe also is graphics API for special graphics computers that use Mutlpipe processing for advanced graphics.

References

- <http://www.sgi.com/software/opengl/>
- <http://www.gamers.org/dEngine/xf3D/howto/3Dfx-HOWTO-9.html>
- http://www.exaflop.org/docs/d3dogl/d3dogl_jc_plan.html
- http://www.sgi.com/newsroom/press_releases/2002/july/opengl.html
- <http://www.xbitlabs.com/articles/video/display/voodoo3-3000.html>
- <http://translate.google.com/translate?hl=en&sl=de&u=http://www.tommti-systems.com/main-Dateien/reviews/opengldirectx/openglvsdirectx.html&prev=/search%3Fq%3Dopengl%2Bvs%2Bglide%26hl%3Den%26lr%3D%26ie%3DUTF-8%26oe%3DUTF-8%26sa%3DG>
- <http://slashdot.org/articles/99/05/14/195242.shtml>
- <http://www.gamedev.net/reference/articles/article1775.asp>
- <http://www.3dlabs.com/support/developer/ogl2/>
- <http://www.nvnews.net/articles/opengl/introduction.shtml>
- <http://www.exaflop.org/docs/d3dogl/ind.html>
- Direct3D vs. OpenGL, Markus Weissmann, 2002
- http://www.nvnews.net/articles/nick_triantos_interview.shtml
- http://www.nvnews.net/articles/steve_mosher_interview.shtml
- <http://www.geek.com/news/geeknews/2002Jul/gam20020718015469.htm>
- <http://www.gamedev.net/reference/articles/article608.asp>
- <http://www.avsim.com/fsbench/glossary.htm>