Anatomy of a Game

Making a C++ OpenGL Game

C++ and OpenGL

- Quick
- Portable
- Customizable

John Carmack on OpenGL

http://rmitz.org/carmack.on.opengl.html

Application Level

- Create window to display
- Initialize variables
- Load any files we need at the beginning
  - Save files
  - Levels
- Game Loop
  - Update
  - Swap Buffers
  - Wait
- Any post game saving
  - Save progress
- Close Window

Game Loop

- Input
- Update
- Output

Game Loop: Input

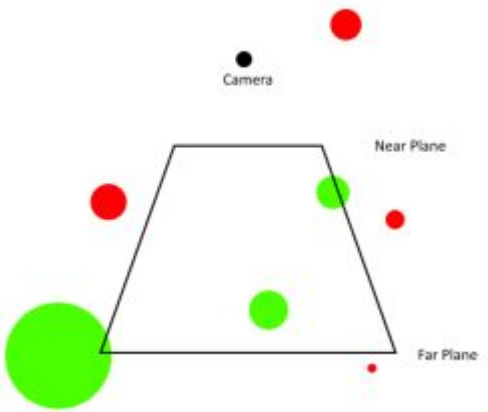
- Keyboard & mouse
- Game controllers
- Joystick
- Microphone
- Camera
- Any other hardware we can access and want to access

Game Loop: Update

- MOVE things around
  - Player, camera, AI
- CHECK conditions and triggers for things to change
  - Explosions, collision, death, victory
- CREATE/REMOVE items due to triggers or conditions
  - IF player is in explosion THEN kill him, play a sound, add a point
- SCRIPTS are run to do custom code injections or handle certain areas of code
  - Usually lua scripts are used (seen in WoW, Skyrim, Roblox)

Game Loop: Update contd

- Build collision data
  - Octrees, Quadtrees, Grids, Hierarchical Grids, Voronoi diagrams, etc.
  - Used for the many checks in a game
- Update graphics variables on the CPU and some GPU
  - What objects need to be drawn in this frame? - View frustum collision query



Game Loop: Output

- Graphics (vision)
- Audio (hearing)
- Rumble (touch)

Rendering

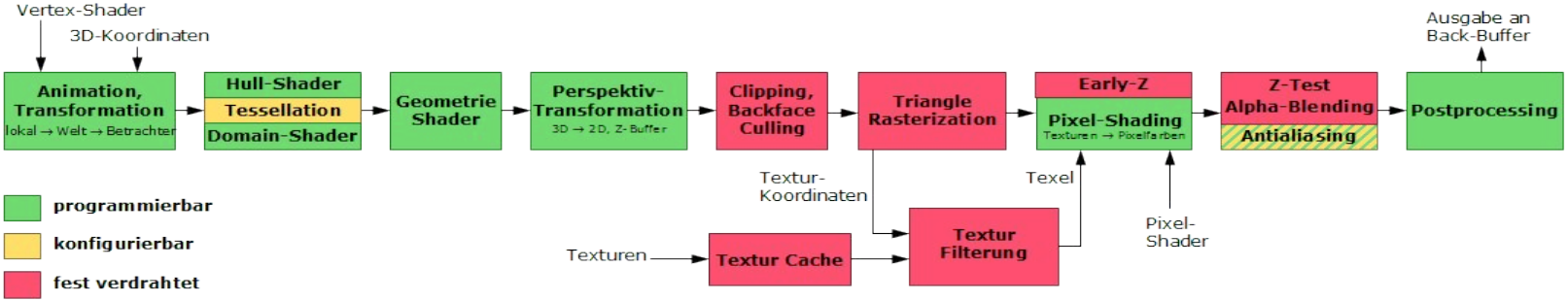
How do we get pixels to look a certain way on the screen?

We use the GPU for most games to render the graphics on screen.

We can get away with using CPU only, but it is harder and slower to achieve a good result.

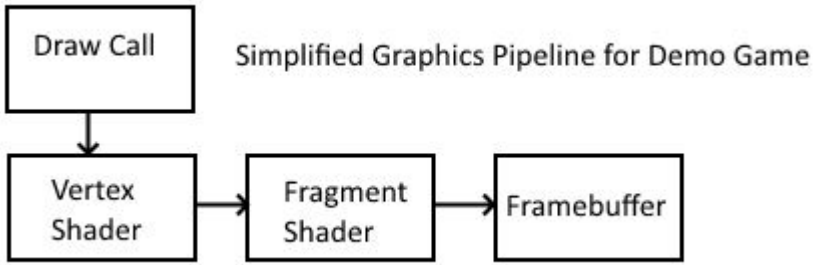
Rendering: Graphics Pipeline

Wikipedia Version



Rendering: Graphics Pipeline Simplified

Our version for our basic game demo



Rendering: Screen Space

- Transform Object or World space object to Screen Space
  - Multiply our position vector by a matrix that is composed of a View and Projection Matrix
  - Done in our Vertex Shader
- Fancy graphics of Screen Space to World Space or Object Space
  - Sometime in our Fragment Shader we want to know the 3D position
  - Can be done by multiplying by the inverse of the matrix that took us to Screen Space

Rendering Misc

- Order Matters
  - We have lots of tools to help, but we must know order of renderings
  - Affects transparency layers
- OpenGL is a state machine so we need to change states in between render calls
  - Need to clear depth buffer, color buffer
  - Enable/Disable depth testing
  - Clear buffers after enabling buffers to be written
  - Switch textures
  - Change bound data

Audio

- Similar process to how we can load data into the GPU
- Copy data from an input source and write it into a buffer to be written out

Rumble

- Dependent on hardware
- Usually writing values into an array of how hard you want the hardware to vibrate at that time interval

Loading/Saving Data

- Best to write future proofed code
  - Headers for each class/object being saved that gives it a unique id and body size
  - If your code doesn’t know a specific header skip to the end of it and continue on
- Need endian checks for code
  - Can be something simple like the first header has an endian byte check
  - Allows cross platform saving
- Save/load on a separate thread
  - Prevents game stuttering
  - Can load in a future area when we get close enough to it
- Checksums
  - Can be helpful for checking corruption and crashes
  - Can plan a data recovery method if need be, or a default value

Operating System Specific Code

- Try to encapsulate all this code
- Don’t spread it throughout the game
- Write wrappers and interfaces
- Saving/Loading can be abstracted out of specific system code

Balancing Load

- Time code in CPU and GPU to look for bottlenecks
- May need to examine GPU code for eliminating loops or extra data bandwidth
- Need to plan how much data we move around and when to prevent stuttering

Timewasters

- Advanced Physics
  - Getting things right is hard, need to know lots of math
- Changing Model data
  - Plan what data you need on each model well in advance
  - Refactoring layers of code to get it to work again is awful
- Realistic Graphics
  - Can be done, just takes lots of time and more time to optimize it to decent framerate
  - Pick a good enough
- Not planning ahead
  - Faster to program a game that you can fully plan out ahead of time

Questions?

Other Resources

Real-Time Collision Detection by Christer Ericson (textbook on most object-object collision algorithms as well as other spatial partitioning algorithms)

Shader Toy https://www.shadertoy.com/ (learn or look at fantastical shader created scenes/simulations)

NeHe Productions https://nehe.gamedev.net/ (graphics tutorials, slightly outdated but some principles are the same)

Presenter Information

Robert Dmytruk

BSci Computing Science UofA 2011

rwdmytruk@gmail.com