CS5625 Interactive Computer Graphics

Steve Marschner
Spring 2016

01 Introduction
Naughty Dog—The Last of Us (Remastered, 2014)
Ubisoft — Child of Light (2014)
Transform vertices to screen coordinates

Find all the pixels covered by the triangle

Fill all the pixels with the triangle’s color
Perform lighting calculations to find vertex colors

Transform vertices to screen coordinates

Find all the pixels covered by the triangle

Fill all unoccluded pixels with the interpolated vertex colors and depth
Perform lighting calculations to find vertex colors
Transform vertices to screen coordinates
Find all the pixels covered by the triangle
Look up a texture map value
Fill all unoccluded pixels with a function of the texture and the interpolated vertex colors, as well as the depth
Perform elaborate lighting calculations to find vertex colors
Transform vertices to screen coordinates
Find all the pixels covered by the triangle
Look up a value from one or more 1D, 2D, or 3D texture maps
Fill all unoccluded pixels with a complicated, adjustable function of the textures and the interpolated vertex colors, as well as the depth
Pixar—*Ratatouille* (2007)
How To Draw a Triangle in 2001

- Execute a vertex program over all the vertices
- Find all the pixels covered by the triangle
- Execute a fragment program over all those pixels
- Fill all unoccluded pixels with the resulting color and depth
Development of Hardware Capabilities

Workstation era
- ’85–’87: transform and render flat-shaded points, lines, polygons (no z buffer)
- ’88–’91: transform, light, and render smooth shaded polygons
- ’92–: transform, light, and render texture-mapped, antialiased polygons

PC era
- ’95–’98: render texture-mapped polygons
- ’99–’00: transform, light, and render texture-mapped, antialiased polygons
- ’01–’06: execute vertex and fragment shaders over antialiased polygons
- ’07–’09: execute vertex, geometry, and fragment shaders over antialiased polygons
- ’10–: execute vertex, geometry, tessellation, and fragment shaders over antialiased polygons
“Moore’s Law” Enabled by Transistor M1 Half Pitch Dimension Technology

Transistor Gate Technology Power-performance Management Enabled by “Equivalent Scaling”
Figure 1. Board-level block diagram of an intermediate configuration with 8 Geometry Engines on the geometry board, 2 raster memory boards, and a display generator board.
SGI InfiniteReality Architecture (1996)
It's one thing to recognize the future potential of a new processing architecture. It's another to build a market before that potential can be achieved. There were attempts to build chip-scale parallel processors in the 1990s, but the limited transistor budgets in those days favored more sophisticated single-core designs.

The real path toward GPU computing began, not with GPUs, but with non-programmable 3G-graphics accelerators. Multi-chip 3D rendering engines were developed by multiple companies starting in the 1980s, but by the mid-1990s it became possible to integrate all the essential elements onto a single chip. From 1994 to 2001, these chips progressed from the simplest pixel-drawing functions to implementing the full 3D pipeline: transforms, lighting, rasterization, texturing, depth testing, and display.

NVIDIA's GeForce 3 in 2001 introduced programmable pixel shading to the consumer market. The programmability of this chip was very limited, but later GeForce products became more flexible and faster, adding separate programmable engines for vertex and geometry shading. This evolution culminated in the GeForce 7800, shown in Figure 3.

Figure 3. The GeForce 7800 had three kinds of programmable engines for different stages of the 3D pipeline plus several additional stages of configurable and fixed-function logic.

(Source: NVIDIA)
So-called general-purpose GPU (GPGPU) programming evolved as a way to perform non-graphics processing on these graphics-optimized architectures, typically by running carefully crafted shader code against data presented as vertex or texture information and retrieving the results from a later stage in the pipeline. Though sometimes awkward, GPGPU programming showed great promise.

Managing three different programmable engines in a single 3D pipeline led to unpredictable bottlenecks; too much effort went into balancing the throughput of each stage. In 2006, NVIDIA introduced the GeForce 8800, as Figure 4 shows. This design featured a “unified shader architecture” with 128 processing elements distributed among eight shader cores. Each shader core could be assigned to any shader task, eliminating the need for stage-by-stage balancing and greatly improving overall performance.

The 8800 also introduced CUDA, the industry's first C-based development environment for GPUs. (CUDA originally stood for “Compute Unified Device Architecture,” but the longer name is no longer spelled out.) CUDA delivered an easier and more effective programming model than earlier GPGPU approaches.
Like Fermi, Kepler GPUs are composed of different configurations of Graphics Processing Clusters (GPCs), Streaming Multiprocessors (SMs), and memory controllers. The GeForce GTX 680 GPU consists of four GPCs, eight next-generation Streaming Multiprocessors (SMX), and four memory controllers.
Like Fermi and Kepler, GM204 is composed of an array of Graphics Processing Clusters (GPCs), Streaming Multiprocessors (SMs), and memory controllers. GM204 consists of four GPCs, 16 Maxwell SMs (SMM), and four memory controllers. GeForce GTX 980 uses the full complement of these architectural components (if you are not well versed in these structures, we suggest you first read the Kepler and Fermi whitepapers).

Another version of the chip, with 13 SMs, will ship concurrently and be called GeForce GTX 970. In the future we plan to offer additional products based on GM204 that will ship with different combinations of GPCs, SMs, and memory controllers to address various segments of the graphics market.

The memory subsystem has also been significantly revamped. GTX 980's memory clock is over 15% higher than GTX 680, and GM204's cache is larger and more efficient than Kepler's design, reducing the number of memory requests that have to be made to DRAM. Improvements in our implementation of memory compression provide a further benefit in reducing DRAM traffic—effectively amplifying the raw DRAM bandwidth in the system.

Maxwell Streaming Multiprocessor

The SM is the heart of our GPUs. Almost every operation flows through the SM at some point in the rendering pipeline. Maxwell GPUs feature a new SM that's designed to provide dramatically improved performance per watt than prior GeForce GPUs. Compared to GPUs based on our Kepler architecture, Maxwell's new SMM design has been reconfigured to improve efficiency. Each SMM contains four warp schedulers, and each warp scheduler is capable of dispatching two instructions per warp every clock. Compared to Kepler's scheduling logic, we've integrated a number of improvements in the scheduler to further reduce redundant re-computation of scheduling decisions, improving energy efficiency. We've also integrated a completely new datapath organization. Whereas Kepler's SM shipped with 192 CUDA Cores—a power-of-two organization—the Maxwell SMM is partitioned into four distinct 32-CUDA core processing blocks (128 CUDA cores total per SM), each with its own dedicated resources for scheduling and instruction buffering. This new configuration in Maxwell aligns with warp size, making it easier to utilize efficiently and saving area.

Figure 3: GM204 SM Diagram (GM204 also features 4 DP units per SMM, which are not depicted on this diagram)
What is Graphics?

• Generating images!

• Create them
  – Modeling
  – Animation
  – Rendering

• Manipulate them
  – Imaging
What is in 5625?

- Generating images
- Modern Graphics Pipeline
- Create them
  - Modeling
  - Animation
  - Rendering
- Manipulate them
  - Imaging
- Focus on Interactive Graphics
Modeling
Subdivision Surfaces
LOD: Level-of-Detail
LOD: Level-of-Detail
Animation
Morph Targets

José Alves da Silva—Corlyorn Family (Vodafone campaign)
Rendering
Rob Cook’s vases

Source: Cook, Torrance 1981
Texture Mapping

- Bump Maps
- Normal Maps
- Environment Maps
- Irradiance Maps
- ....
Displacement Maps

Max Displace 1.5Mil  Normal Map 2900Tris  Wire
Filtered Environment Mapping

- Environment map $\rightarrow$ radiance
- Filter this map $\rightarrow$ irradiance (diffuse lighting)
- Fast diffuse and ambient (just a lookup, or eqn)
Anti Aliasing of TMs and Theory of Sampling
Shadow Algorithms

• Crucial for spatial and depth perception
Shadow Maps

• Introduced by Lance Williams (SIGGRAPH 1978)
• Render scene from light’s view
  – black is far, white is close
Shadow Volumes

- Clever counting method using stencil buffer
- Can cast shadows onto curved surfaces
Soft Shadows

- Soft shadows appear more natural
  - Hard to get soft shadows in hardware
Ambient Occlusion
Imaging
Modeling flare in a camera lens

Hullin et al. SIGGRAPH 2011
Tone reproduction operator

visual model

scene observer model

display observer model

display device model

scene radiances

perceptual match

display inputs

display device

Kavita Bala, Cornell University
Modeling flare in the eye

Figure 14: Two highway scenes before and after the scotopic glare algorithm. The orientation of the headlights is made obvious by the degree of glare.

Figure 15: An indoor simulation before and after the mesopic glare algorithm.

Figure 16: The Sun showing through leaves before and after the photopic glare algorithm. The location of the Sun is obvious only after the glare is added. Note that there is no lenticular halo because the pupil of the viewer is contracted.

Greger et al. SIGGRAPH 2005
Projects
CS 5625 Coursework

8 mini-assignments (probably in pairs)
- mostly implementation, some written
- Primarily Java and OpenGL
- anticipated topics: shading, texturing, shadow, subdivision surfaces, antialiasing/filtering

Midterm exam

Final project (groups of 2–4)
- project proposal
- mid project evaluation
- final project demos, presentations, writeup
Mesh animation
Texture antialiasing
Reflection mapping
Deferred shading
Shadow Mapping
Soft shadows
Final project examples

Spring 2015
- MOTION BLUR
- LENS FLARE
Fight your way through the zombie fairies to rescue Orin!

Ari Karo, Christopher Yu, Jonathan Behrens, Jeremy Cytryn | Subterranean Arsonism
About CS5625

Prereqs

• introductory graphics course (e.g. 4620) or instructor permission

Dissemination

• website www.cs.cornell.edu/Courses/cs5625
  - schedule (very much subject to change!)
  - announcements, updates

• CMS
  - homeworks, lecture notes

• Piazza
  - discussion, questions
Academic Integrity

Don’t copy code from Web without careful attribution
  · small snippets of, e.g. OpenGL boilerplate OK with attribution

Collaboration only when projects/homeworks are with groups

Lots of detailed discussion is not ok
  · need to come up with answers as separate groups/individuals

Always cite sources of code and ideas
  · think carefully about who and what contributed to your work
  · if you tell me what is going on, there is no AI problem
Recommended texts

**Real-time Rendering**
- Akenine-Moller, Haines, Hoffman
  - available via library
  - not required

**Other books**
- many listed on website

**Online resources**
- lots of them!
Other Policies

Late policy

• 5 free late days over the semester
  - in total over all assignments
  - not for the final project

• Otherwise, 10% penalty per day for up to 5 days
  - after that, 0
Final project

An interactive 3D game with fancy graphics

Open ended, needs to have technically impressive results

Ways to impress

- rendering: shading, shadows, global illumination, …
- modeling: splines, subdivision surfaces, procedural generation, …
- animation: particle systems, character motion, collision detection, …
- imaging: flare, antialiasing, …

Focus is on graphics, not gameplay