ZJNU

# Introduction to the Module

Computer Graphics with Vulkan - ZJNU0101600533

ZHEJIANG NORMAL UNIVERSITY

# Outline

ZJNU

# Aims and Objectives

ZJNU

Aims:

- Understand how to build 3D geometric objects
- Understand graphics rendering techniques
- Understand underpinning mathematics for working in 3D rendering

Objectives:

- Develop a scene using 3D geometric objects
- Utilise graphics rendering techniques within a scene
- Utilise mathematical techniques to develop a scene using rendering techniqes

# Organisation

ZJNU

One lectures per week:

- **See Timetable**

One practical per week
The practical sessions are hands-on labs of the concepts discussed in the lectures.
One tutorial per week
The tutorial will cover mathematical concepts required for the module.
Assessment:

- Coursework - create scenes in Vulkan
- Exam - maths, graphics pipeline, rendering concepts

# Contact

ZJNU

Questions/Issues

- Benjamin Kenwright
- email: bkenwright@ieee.org

- Open Door Policy
- Problems/Help
- Within Reason

# Ongoing Assessment

ZJNU

Week 1:

- Recommended Texts/Exercises
- Class Quizzes/Discussion
- GitHub
- Website/Blog (email me the URL)

Week 2:

- LaTeX Report (show in Lab)
- Tutorial 1 Quiz

Week 3:

- ...

# Example

ZJNU

Deliverables...

- peer review
- videos
- blog/github
- technical reports
- presentations
- ...

Your work will be demonstrated to the entire class/university. Demonstrate high quality competence of computer graphics and software engineering (e.g., ability to write clear well formatted graphics code).

# Graphics Rendering

ZJNU

Graphics rendering is the use of computational techniques to produce 3D scenes (usually by some form of description file)

Graphics rendering is a major topic in the area of 3D computer graphics. Other concepts such as animation also fall into the 3D computer graphics area.

From our point of view, we are concerned with just creating a 2D image (the computer screen) from the information used to describe a 3D scene. This module will not concern itself with animation beyond the basic idea of transforming solid objects.

Computer graphics were originally developed at Boeing to support GUI development for pilots (in 1961).

# History - Future World

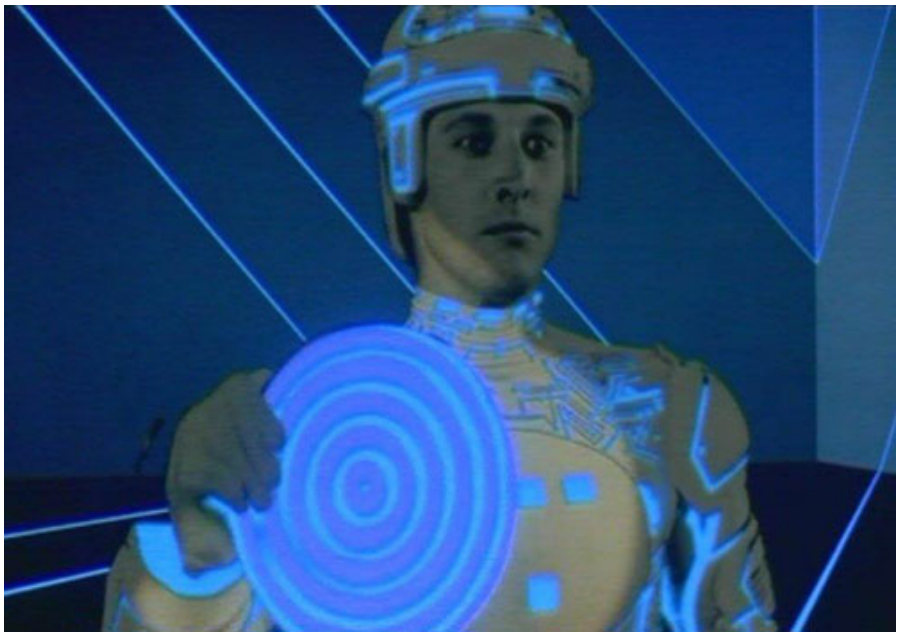
ZJNU

Future World (1976) was the first film to feature computer generated graphics. The animated face (left) is simple by today's standards, but was cutting edge in it's day.

The face was the work of Ed Catmull and Fred Parke. You will come across the name Catmull quite often in computer graphics (he worked at Lucasfilm, the division which became Pixar, where Catmull is the current president).

# History - Tron

ZJNU

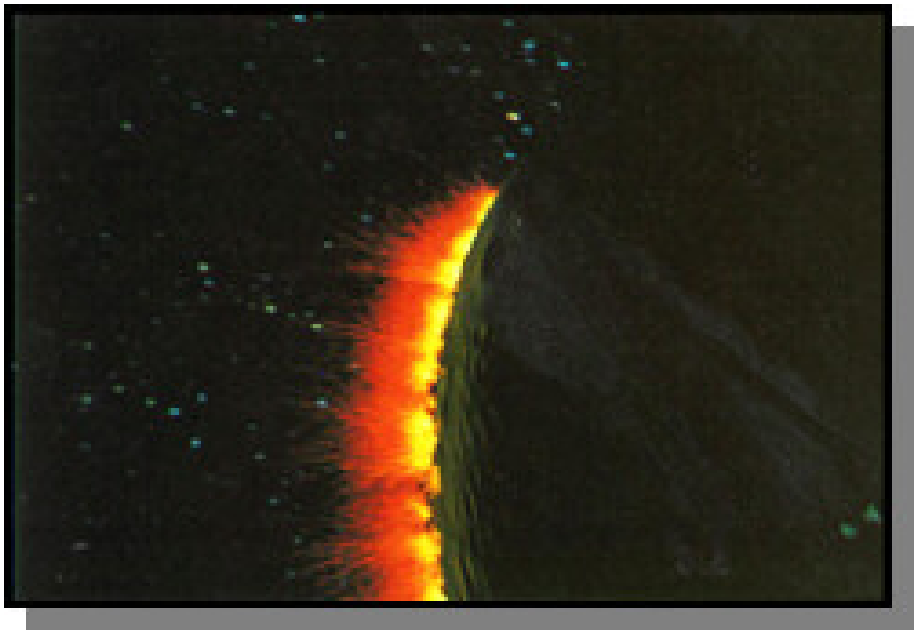


Tron (1982) was the first film to make extensive use of computer animation techniques. The computer used to create the generated effects in Tron had only 2MB of memory, and 330MB of storage.

One innovation from Tron was the development of Perlin noise by Ken Perlin. This has formed the basis for many procedural content techniques.

# History - Wrath of Khan

ZJNU



Star Trek 2: The Wrath of Khan (also 1982) featured the first particle effects in a movie. Particle effects are used for many fuzzy object effects, such as fire, smoke, and explosions.

Particle systems are also used for effects such as grass, hair, cloth, and other concepts. No one really blows anything up anymore, but rather use a particle effect instead.

# History - Toy Story

ZJNU


Toy Story (1995) was the movie that changed everything in computer graphics rendering and animation in movies. Toy Story is the first fully computer animated movie.
Each frame of the final movie took between 45 minutes and 30 hours to render (well outwith the real-time requirements of games).
One of the key features of the process was Pixar's RenderMan shader language. Shader languages are fundamental to modern 3D graphics rendering.

# History - Star Wars Episode I

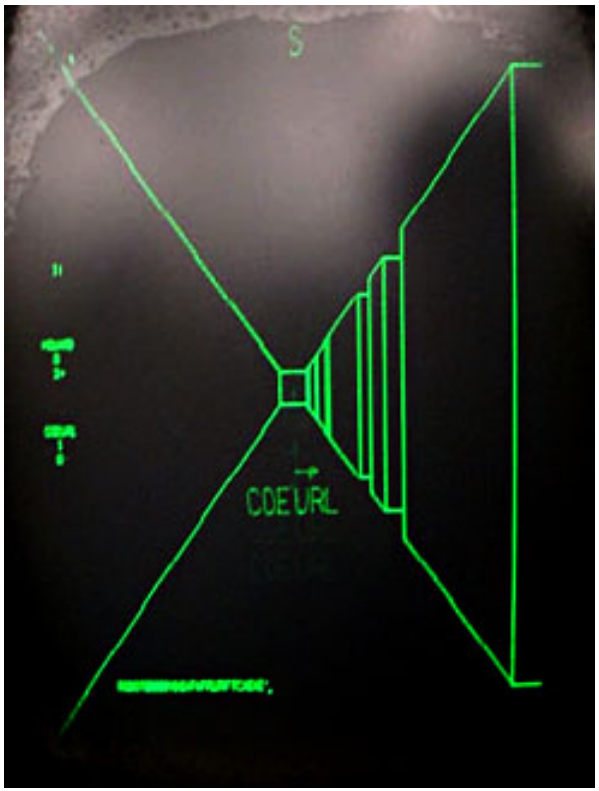
ZJNU

Star Wars: Episode I (1999) was the first live action film to use extensive digital effects, especially the use of completely rendered characters, such as Jar Jar Binks.

Today we don't even notice the effects that appear in films, but it is only recently that we have been able to achieve realistic computer generated effects within live action movies.

# History - Maze War

ZJNU



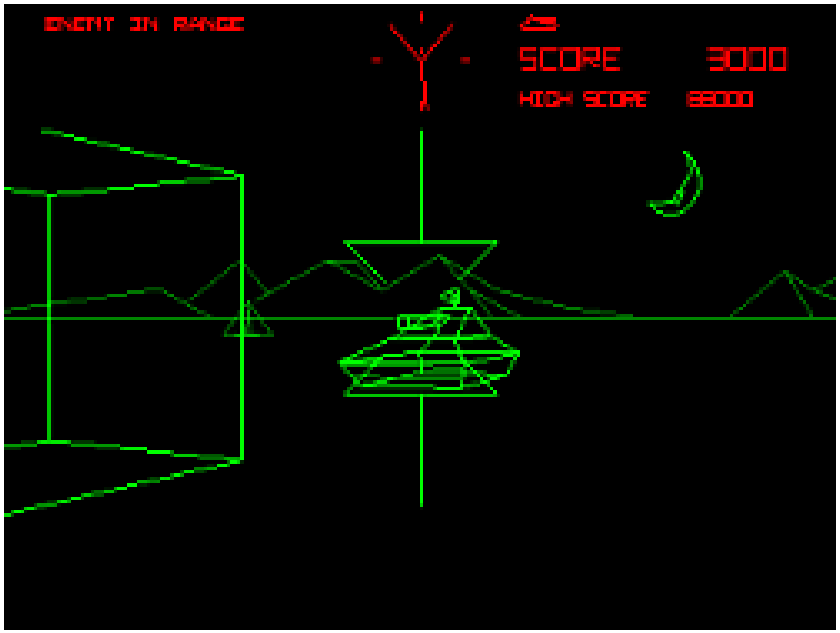

In parallel with work in movies, video games have also been working towards realistic 3D scenes using computer graphics techniques.

Unlike movies, games have to operate in real-time (an image has to be generated for a user - a minimum of 24fps is required).

Maze War (1974) is the first example of a first person style game. Compare Maze War to something like Call of Duty to see how much has changed.

# History - Battlezone

ZJNU




Battlezone (1980) was the first game considered to have a true 3D world to explore. Unlike Maze War, the player had freedom of movement in 3D space.

Battlezone also used a headset in the arcade version. It is therefore also considered the first virtual reality game as well.

# History - Elite

ZJNU


Elite (1984) brought a 3D universe to explore. Elite is considered one of the cornerstones of using procedural techniques to develop an entire universe to explore, on hardware that had less than 100KB of memory. Developed by David Braben and Ian Bell while they were at the University of Cambridge. David Braben currently fronts Frontier Games (Elite 2 was subtitled Frontier), which developed Roller Coaster Tycoon 3. David is also one of the people behind the Raspberry Pi.

# History - Wolfenstein 3D

ZJNU


Wolfenstein 3D (1992) essentially created the FPS genre as we know it.

Wolfenstein 3D did not have fully rendered 3D graphics as we know it today as this was before hardware support for 3D in personal computers.

Wolfenstein 3D instead used 2D images and traditional approaches for most of the graphics, but utilised a 3D space and some texturing techniques.

# History - Descent

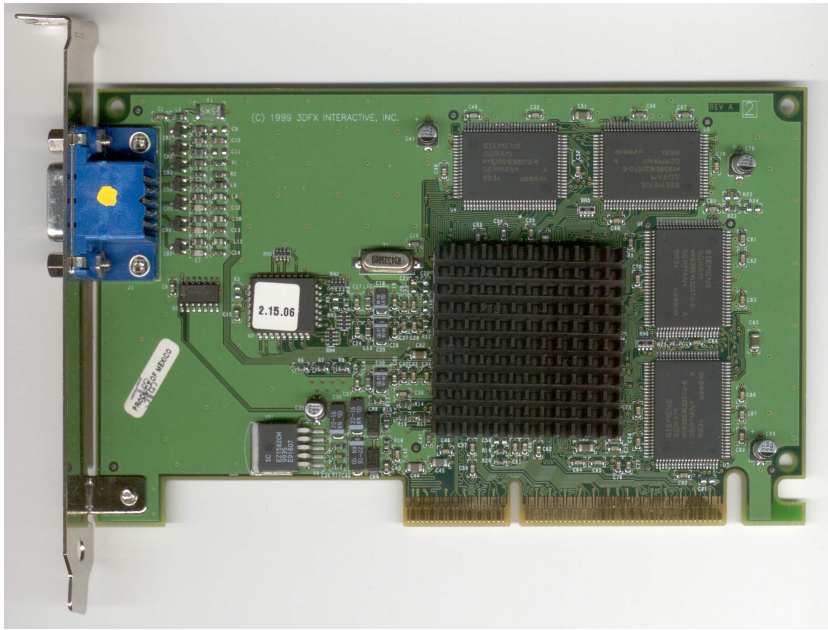
ZJNU


Descent (1995) can be considered the first game to use modern techniques. The use of polygons and texturing is apparent in Descent.

The graphical effects in Descent were made possible by the release of the first commercial 3D accelerator aimed at the personal computer market. The chipset was Voodoo, and it was released by 3Dfx Interactive.

# History - 3Dfx Interactive

ZJNU



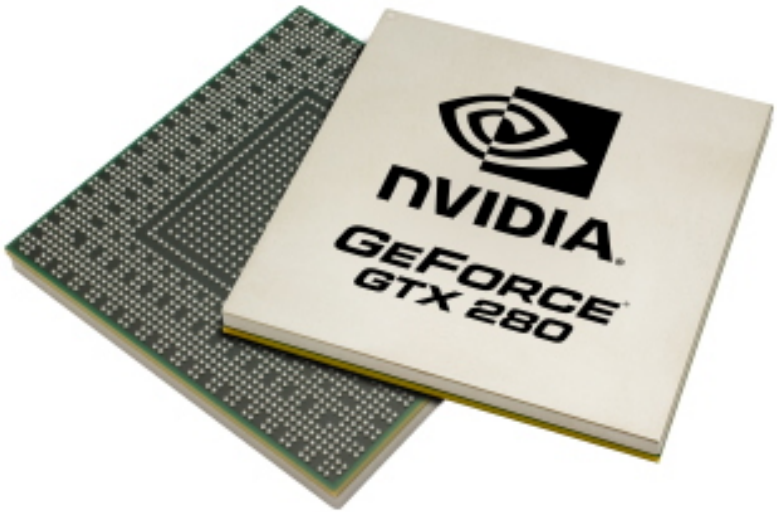

3Dfx Interactive launched t home use 3D accelerator card in 1996 - due mainly to a reduction in the cost of memory.

The first cards only had around 4MB of RAM, and had a clock processor of around 50MHz. Simply having a graphics co-processor though changed everything.

3Dfx Interactive filed for bankruptcy in 2002. Most of its intellectual property and staff were taken by Nvidia.

# History - Nvidia and the GPU

ZJNU




Nvidia coined the phrase Graphics Processing Unit (GPU) in 1999. The advent of GPUs was the point where modern graphical rendering techniques were born.

The GPU brought around the concept of texturing and lighting to the consumer market, and therefore games. By providing hardware accelerated technology for 3D rendering, better visual effects than ever before were possible.

The basic graphical processing concepts are still in current generation hardware.

# Current Trends

ZJNU

Current trends in graphics rendering techniques focus on a mix of better realism (specifically for movies), and better performance (particularly for games).

In this module we will be focusing on techniques more suited for real-time rendering techniques. Although there are techniques which are quite modern, most of our work will actually involve techniques first developed in the 1970s (most game based graphics rendering does this).

The main driver behind games based graphics rendering is the limitations of current generation hardware.

# Previous Generation Hardware

ZJNU

XBox 360 specs:

- Triple core 3.2 GHz processor
- 512MB of memory, 10MB of graphics memory
- Only DirectX 9 compatible

PS3 specs:

- Cell Processor (can be thought of as multi-core)
- 512MB of memory (256MB for graphics). No dedicated (on board) graphics memory
- OpenGL ES 2.0 compatible

Wii specs:

- 729 MHz processor
- 88MB of memory, 3MB of graphics memory
- No programmable shader support

# Current Generation Hardware

ZJNU

XBox One specs:

- Eight core 1.75 GHz processor
- 8GB DDR3 memory, 32MB of graphics memory
- 853 MHz GPU, 768 shader cores
- DirectX 11.1 compatible

PS4 specs:

- Eight core 1.6 GHz processor (supposedly 2.75 GHz capable)
- 8GB GDDR5 memory. No dedicated (on board) graphics memory
- 800 MHz GPU, 1152 shader cores
- OpenGL ES 3.0 compatible

Wii specs:

- Triple core 1.24 GHz processor
- 2GB DDR3 memory, 32MB of graphics memory
- 550 MHz GPU, 400 shader cores (rumoured)
- OpenGL ES 3.0 compatible?

# Programmable Shaders

ZJNU

Currently, most work in modern real-time rendering focuses on utilising the GPU to produce more realistic effects (typically photo-realism).

Shader programs are small pieces of code specifically developed to run on the GPU to produce a certain graphical effect. The GPU is designed to handle graphics rendering using our current approach (e.g. polygon based rendering).

More recently, GPUs have enabled general purpose programming through toolkits such as Nvidia's CUDA and OpenCL. DirectX, OpenGL and Vulkan support compute shaders which enable programmers to utilise the GPU for processes not specifically rendering based.

# Photo-realism

ZJNU

The goal for graphics rendering is to achieve photo-realism. This means creating an image that is indistinguishable from a photograph.

There are already a number of videos reportedly from next-gen engines that support photo-realism - cost of development will likely be a hindrance.

Realism does lead to the üncanny valleyëffect.

# Areas

ZJNU

There are a number of core areas where graphics rendering is useful:

- Computer Aided Design (CAD)
- Computer Generated Images (CGI)
- Games

Although most of the ideas we will discuss in the module are applicable to all areas, we will concern ourselves with ”real-time”rendering, which is what we are interested in from a games point of view.

# Mathematical Understanding

ZJNU

We will also use graphics programming / rendering to better understand 3D space and geometry.

We will be covering quite a few ideas in the module, starting from the basics of a geometry system (Euclidean / formal geometry), working through vectors and transformations,

For example, at a minimum, you should feel comfortable with the following:

$$\vec{p} \cdot \vec{q} = \sum_{i=0}^{n} p_i q_i$$

# Graphics Cards

ZJNU

Graphics programming actually requires us to work on two levels:

- CPU and application level - where we set up geometry and send commands for how to render it.
- GPU - where we store geometric data and interpret commands sent from the CPU to render the scene.

We need to be able to understand the difference between the responsibilities of the CPU and the GPU, particularly with modern approaches to graphics rendering.

Initially we will work mainly on the CPU, before moving to the GPU for most rendering operations.

# Operations

ZJNU

The CPU is responsible for building a 3D scene / model:

- A scene consists of a number of models
- A model consists of a number of polygons, textures, etc.
- A polygon consists of a number of points (vertexes), which contain position, normal data, texture coordinates, etc.

We used to send this information to the graphics system to render our scene every frame.

We now store the information on the graphics card, and tell it how to use the data to render a scene.

The number of render images a GPU can produce per second is the ”frame rate”. However, the monitor determines how many actual images are shown per second (usually between 60 and 75).

# Communication with the GPU

ZJNU

In order to communicate with the GPU, we need to use an API:

- OpenGL (or Vulkan)
- DirectX (or specifically Direct3D)

Each API essentially does the same job. However OpenGL and Vulkan is aimed at general purpose graphics rendering. DirectX is a aimed at games development, with Direct3D being the rendering part. Direct3D is still more aimed at the games market than OpenGL/Vulkan.

OpenGL/Vulkan is supported on almost all hardware (except the Xbox) - including iOS and Android. DirectX is only supported on Windows PCs and the Xbox.

# Modernisation

ZJNU

Reasons to keep the API:

- Programmers want a stable platform.
- Programmers want to write one program that will work on all hardware.
- Game companies want the widest market.
- GPU manufacturers want backwards compatibility.

Reasons to update the API:

- New features can be implemented.
- GPU vendors can take advantage of hardware features, and exploit the market.
- Historical features can be a hindrance.

# What is Vulkan?

ZJNU

Vulkan is a low-level API specifically developed for working with 3 graphics - although recently work has focused on other aspects, such as, compute operations (GPU for calculations other than graphics, like simulations and solving mathematical problems)

Vulkan has a number of properties that make it “useful”:

- Low level
- Minimal
- Does not attempt to be convenient
- Fast and efficient

Currently we are at Vulkan 1.0, which is attempting to gain closer interaction with the underlying hardware.

# Primitives and Low Level Components

ZJNU

Vulkan works with the following low level geometric primitives:

- Points
- Lines
- Triangles
- Quads
- Polygons

Vulkan also supports concepts such as:

- Colours
- Normals
- Texture Coordinates

Vulkan utilises these values to produce a 3D scene.

# Shaders

ZJNU

Vulkan supports programmable shaders to enable specific graphical effects.

Vulkan utilises a bytecode format is called SPIR-V. GLSL (GL Shader Language) shader language is able to be compiled to this format. There are also libraries available to support other shader languages. The most commonly used other shader language is Nvidia's Cg.

We will be working with GLSL extensively as we continue through the module. We will also be looking at different shader types.

# Native Applications

ZJNU

We will be writing native Vulkan applications (from the ground-up) including base-classes for Vectors and Matrices to demonstrate an understanding of raw mathematical concepts.

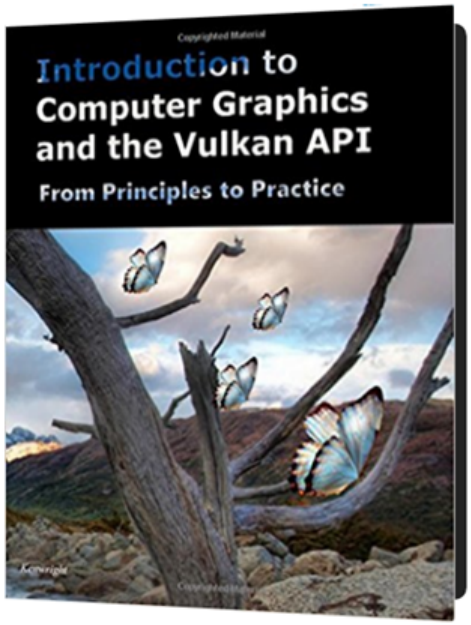
While writing libraries from scratch is tedious and time consuming - it enables you to see the underlying concepts close up (instead of ignoring them). We'll discuss and employ a number of libraries later on when we start to expand our knowledge and implement more advanced features (e.g., model loading libraries, physics libraries, compressions libraries).

We aim to develop solutions that are cross platform, as is Vulkan, meaning our applications will work on any desktop/mobile platform (Linux, Windows and Android).

We'll also discuss input management, such as, keyboard and mouse support - which is essential for user interaction.

# Recommended Reading

ZJNU




Introduction to Computer Graphics and the Vulkan API by Kenwright

Covers all the major aspects of the module (e.g., introduction to graphical principles and the Vulkan API through to graphical effects like lighting and displacement mapping)

Read Chapters 1 & 2 This Week

# Summary

ZJNU

What you should have learnt from this lecture:

- Graphics rendering has a long history, starting in the 1960s at Boeing, through to modern real-time techniques in modern games.
- Graphics rendering can support a number of areas - we will be focusing on real-time aspects and games.
- Current generation hardware dictates the current quality of computer graphics in games.
- Shaders and GPGPU are important ongoing directions
- There are fundamental theoretical reasons we perform computer graphics in the way that we do.
- Core concepts of Vulkan, OpenGL and DirectX have lots of common properties/overlap - Vulkan is the focus of this module (low-level API)

# Questions/Discussion

ZJNU