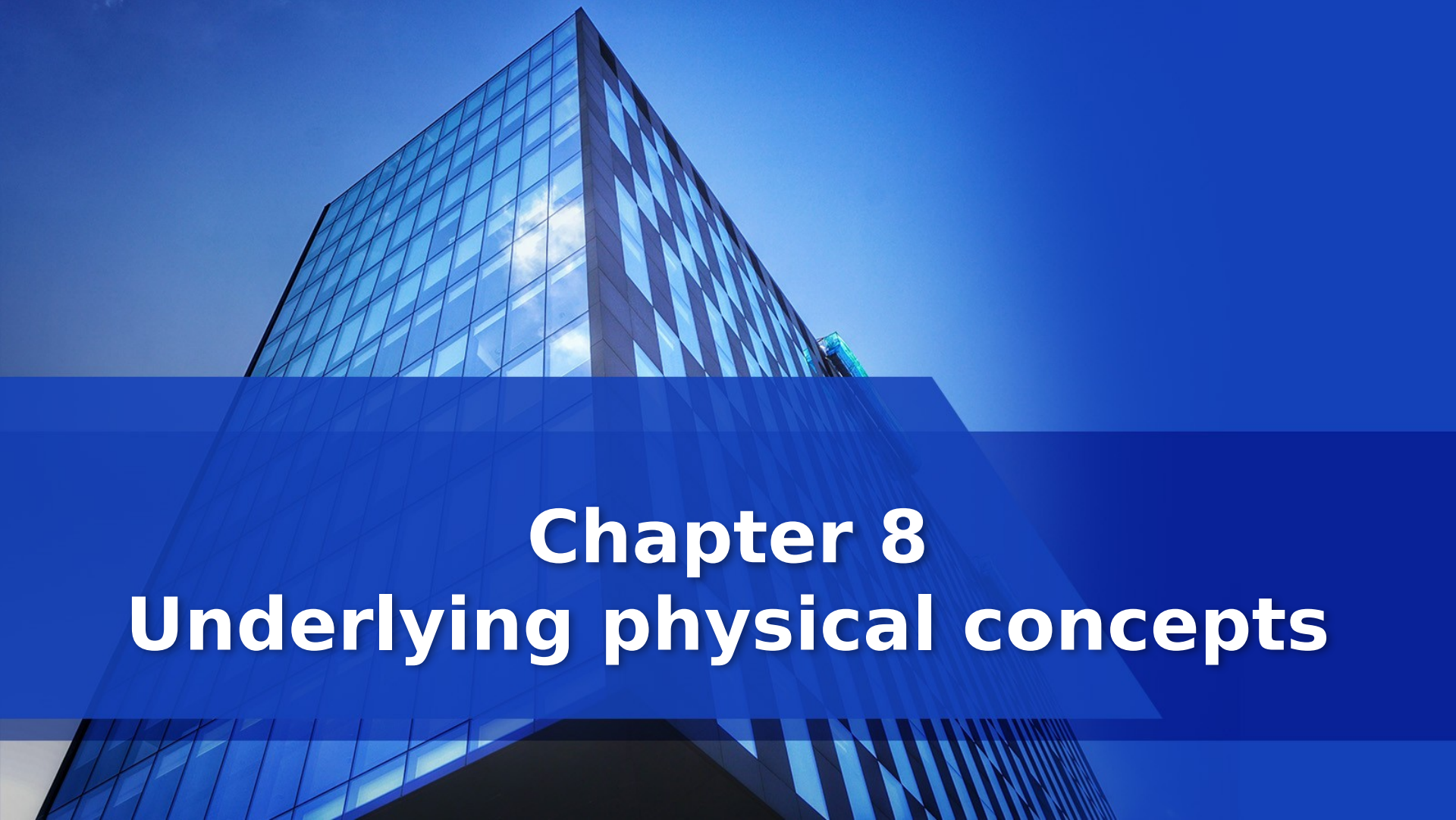


Chapter 8
Underlying physical concepts

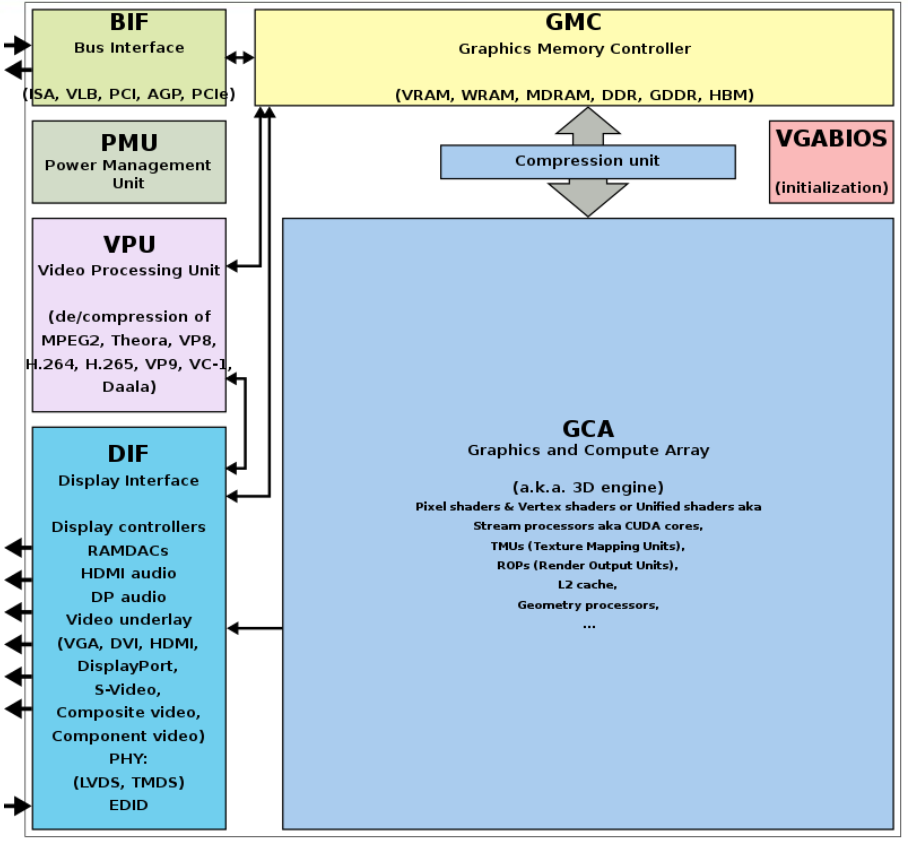
Contents

Underlying physical concepts and their practical implementation based on a 3D game engine environment

1. **Graphics processing unit,**
2. **OpenGL**
3. **Output interfaces**
4. **Game physics**
5. **Game engine**
6. **Components**
7. **Game engines as an industry**
8. **Game middleware**
9. **3 Dimensional game development**
10. **3D game pipeline**
11. **Cloud gaming**
12. **Video game console**

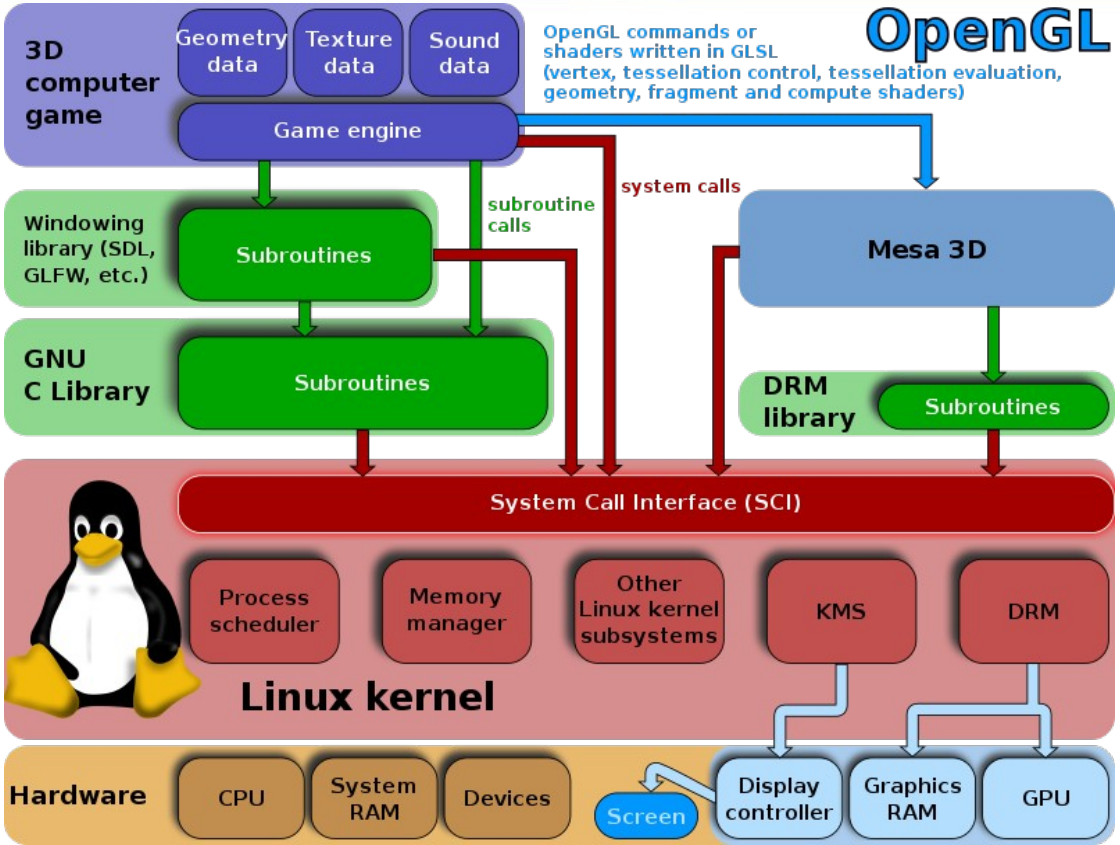
Graphics processing unit & components

- A graphics processing unit (GPU) is a specialized electronic circuit designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display device.
- GPUs are used in embedded systems, mobile phones, personal computers, workstations, and game consoles.
- Modern GPUs are very efficient at manipulating computer graphics and image processing.



OpenGL

- OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics.
- The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering.
- Video games outsource real-time rendering calculations to the GPU over OpenGL.
- The rendered results are not sent back to main memory, but to the framebuffer of video memory instead.
- The display controller will then send this data to the display device.



Output interfaces

- The render output unit, often abbreviated as "ROP", and sometimes called raster operations pipeline, is a hardware component in modern graphics processing units (GPUs) and one of the final steps in the rendering process of modern graphics cards.
- The pixel pipelines take pixel (each pixel is a dimensionless point), and texel information and process it, via specific matrix and vector operations, into a final pixel or depth value.
- This process is called rasterization.

Game physics

- **Computer animation physics** or **game physics** involves the introduction of the laws of physics into a simulation or game engine, particularly in 3D computer graphics, for the purpose of making the effects appear more realistic to the observer.
- There are several elements that form components of simulation physics including the physics engine, program code that is used to simulate Newtonian physics within the environment
- Collision detection, used to solve the problem of determining when any two or more physical objects in the environment cross each other's path.

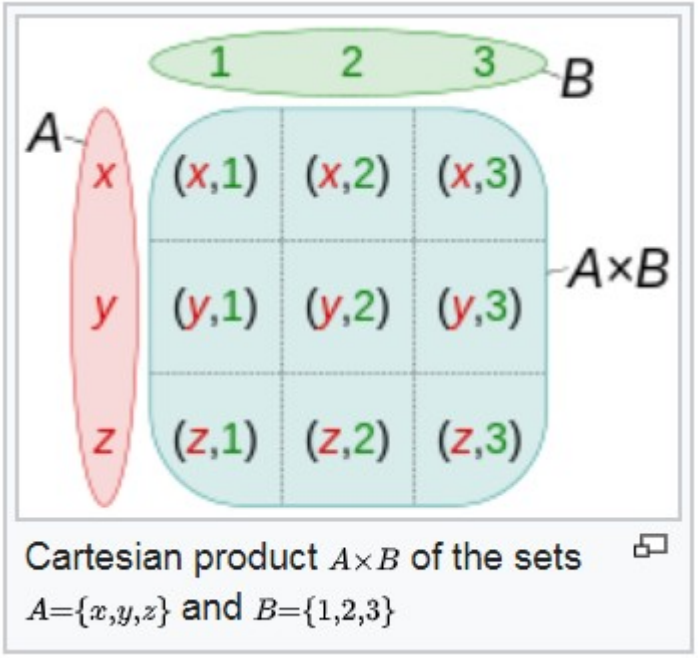
- **Collision detection** is the computational problem of detecting the intersection of two or more objects.
- Collision detection algorithms can be divided into operating on 2D and 3D objects.
- A **hitbox** is an invisible shape commonly used in video games for real-time collision detection; it is a type of bounding box.
- It is often a rectangle (in 2D games) or cuboid (in 3D) that is attached to and follows a point on a visible object (such as a model or a sprite).

- Circular or spheroidial shapes are also common, though they are still most often called "boxes".
- It is common for animated objects to have hitboxes attached to each moving part to ensure accuracy during motion.
- Hitboxes are used to detect "one-way" collisions such as a character being hit by a punch or a bullet.
- They are unsuitable for the detection of collisions with feedback (e.g. bumping into a wall) due to the difficulty experienced by both humans and AI in managing a hitbox's ever-changing locations; these sorts of collisions are typically handled with much simpler axis-aligned bounding boxes instead.

- The axis-aligned minimum bounding box (or **AABB**) for a given point set is its minimum bounding box subject to the constraint that the edges of the box are parallel to the (Cartesian) coordinate axes.
- It is the Cartesian product of *N* intervals each of which is defined by the minimal and maximal value of the corresponding coordinate for the points in *S*.
- The **Cartesian product** of two sets *A* and *B*, denoted $A \times B$, is the set of all ordered pairs (a, b) where *a* is in *A* and *b* is in *B*.
- In terms of set-builder notation, that is

$$A \times B = \{ (a, b) \mid a \in A \text{ and } b \in B \}$$

- A table can be created by taking the Cartesian product of a set of rows and a set of columns.
- If the Cartesian product *rows* × *columns* is taken, the cells of the table contain ordered pairs of the form (row value, column value).



Cartesian product $A \times B$ of the sets $A=\{x,y,z\}$ and $B=\{1,2,3\}$

- Players may use the term "hitbox" to refer to these types of interactions regardless.
- A **hurtbox** is a related term, used to differentiate "object that deals damage" from "object that receives damage".
- For example, an attack may only land if the hitbox around an attacker's punch connects with one of the opponent's hurtboxes on their body, while opposing hitboxes colliding may result in the players trading or cancelling blows, and opposing hurtboxes do not interact with each other.
- The term is not standardized across the industry; some games reverse their definitions of "hitbox" and "hurtbox", while others only use "hitbox" for both sides.

- A debug dialogue box in *Gearheads* controlling an object's hitbox
- The hitbox of a *Gearheads* toy, controlled by the above screen

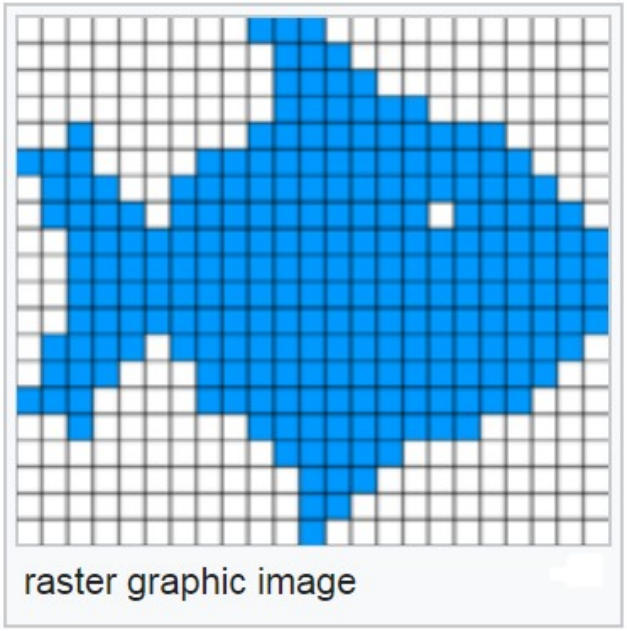




Game engine

- A **game engine**, also known as a **game architecture**, **game framework** or **gameframe**, is a software-development environment designed for people to build video games. Developers use game engines to construct games for consoles, mobile devices, and personal computers.
- The core functionality typically provided by a game engine includes
  - a rendering engine ("renderer") for 2D or 3D graphics,
  - a physics engine or collision detection (and collision response),
  - sound, scripting, animation,
  - artificial intelligence, networking, streaming,
  - memory management, threading, localization support,
  - scene graph, and may include video support for cinematics.

- The rendering engine generates animated 3D graphics by any of a number of methods (rasterization, ray-tracing etc.).
- **Rasterisation** (or **rasterization**) is the task of taking an image described in a vector graphics format (shapes) and converting it into a raster image (a series of pixels, dots or lines, which, when displayed together, create the image which was represented via shapes).


raster graphic image

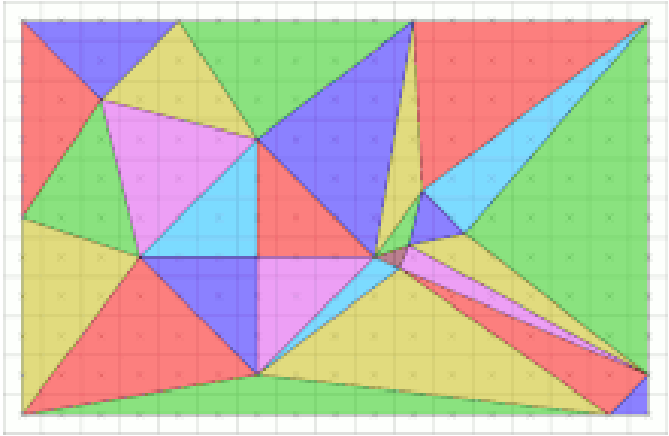
Triangle rasterization

- A common representation of digital 3D models is polygonal.
- Before rasterization, individual polygons are broken down into triangles, therefore a typical problem to solve in 3D rasterization is rasterization of a triangle.
- Properties that are usually required from triangle rasterization algorithms are that rasterizing two adjacent triangles (i.e. those that share an edge) leaves no holes (non-rasterized pixels) between the triangles, so that the rasterized area is completely filled (just as the surface of adjacent triangles).

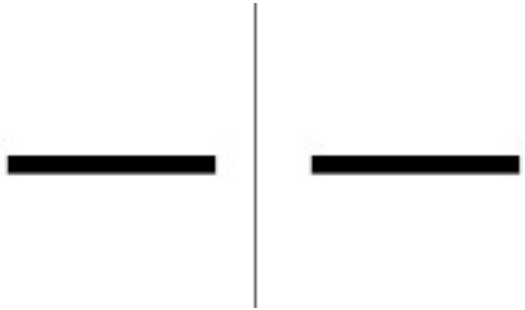


- And no pixel is rasterized more than once, i.e. the rasterized triangles don't overlap. This is to guarantee that the result doesn't depend on the order in which the triangles are rasterized.
- Overdrawing pixels can also mean wasting computing power on pixels that would be overwritten.
- This leads to establishing **rasterization rules** to guarantee the above conditions. One set of such rules is called a **top-left rule**, which states that a pixel is rasterized if and only if
- its center lies completely inside the triangle. Or
- its center lies exactly on the triangle edge (or multiple edges in case of corners) that is (or, in case of corners, all are) either *top* or *left* edge.

- A *top* edge is an edge that is exactly horizontal and lies above other edges, and a *left* edge is a non-horizontal edge that is on the left side of the triangle.
- This rule is implemented e.g. by Direct3D and many OpenGL implementations (even though the specification doesn't define it and only requires a consistent rule.
- In diagram rasterizing triangles using the top-left rule



- The quality of rasterization can be improved by antialiasing, which creates "smooth" edges.
- Sub-pixel precision is a method which takes into account positions on a finer scale than the pixel grid and can produce different results even if the endpoints of a primitive fall into same pixel coordinates, producing smoother movement animations. Simple or older hardware, such as PlayStation 1, lacked sub-pixel precision in 3D rasterization.
- In diagram Pixel (left) vs sub-pixel (right) precision



Components

Main game program

- The actual game logic has to be implemented by some algorithms. It is distinct from any rendering.

Rendering engine

- Instead of being programmed and compiled to be executed on the CPU or GPU directly, most often rendering engines are built upon one or multiple rendering application programming interfaces (APIs), such as Direct3D, OpenGL, or Vulkan which provide a software abstraction of the graphics processing unit (GPU). Low-level libraries such as DirectX, Simple DirectMedia Layer (SDL), and Graphics Library Framework (GLFW) are also commonly used in games as they provide platform-independent access to the windowing system and allow the application to allocate a surface on a computer screen where the graphics will be displayed, this is commonly referred to as graphical context creation.

Audio engine

- The audio engine is the component which consists of algorithms related to the loading, modifying and output of sound through the client's speaker system.
- At a minimum it must be able to load, decompress and play sound files. More advanced audio engines can calculate and produce such things as Doppler effects, echoes, pitch/amplitude adjustments, oscillation, etc.
- It can perform calculations on the CPU, or on a dedicated ASIC. Abstraction APIs, such as OpenAL, SDL audio, XAudio 2, Web Audio, etc. are available.



Physics engine

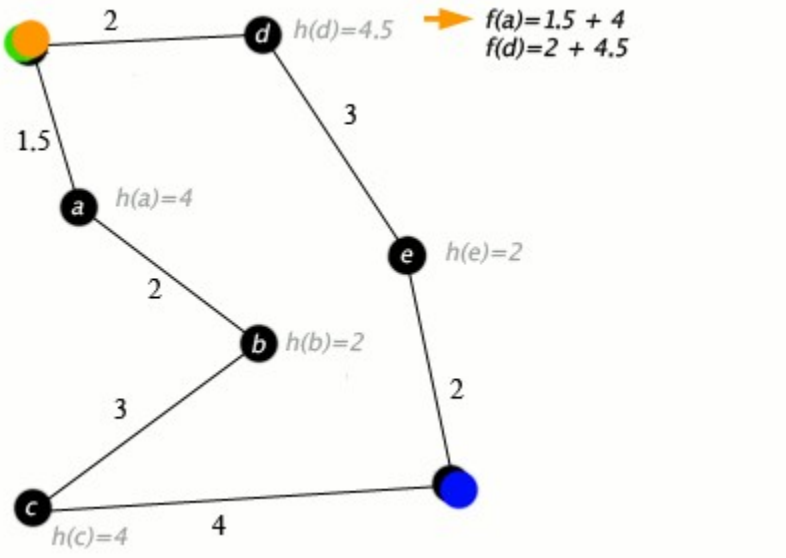
- The physics engine is responsible for simulating the laws of physics realistically within the application.
- Specifically, it provides a set of functions for simulating physical forces and collisions, acting on the various objects within the game at run time.

Artificial intelligence

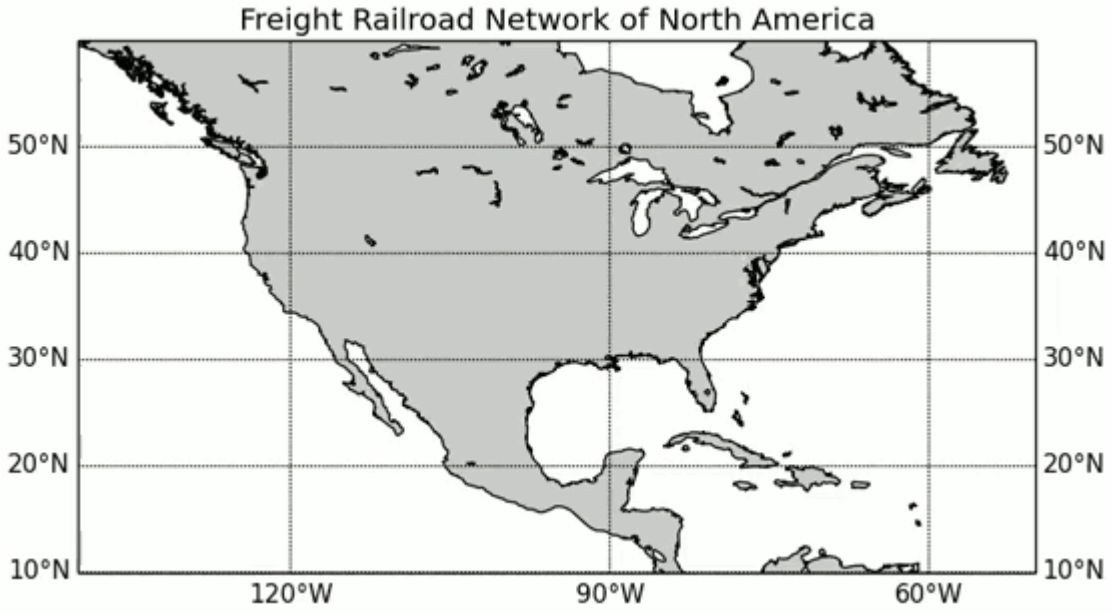
- The AI is usually outsourced from the main game program into a special module to be designed and written by software engineers with specialist knowledge.
- Most games will implement very different AI systems, and thus, AI is considered to be specific to the particular game for which it is created.
- Many modern game engines come packaged with search algorithms such as A-star and subroutines for baking level geometry into a Navmesh which can help speed up the process of scripting AI behavior.

- **A*** (pronounced "A-star") is a graph traversal and path search algorithm, which is often used in many fields of computer science due to its completeness, optimality, and optimal efficiency.
- One major practical drawback is its O(b)^{i}space complexity, as it stores all generated nodes in memory.
- Thus, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, as well as memory-bounded approaches; however, A* is still the best solution in many cases.

- An example of an A* algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point:
- **Key:** green: start; blue: goal; orange: visited
- The A* algorithm also has real-world applications. In this example, edges are railroads and h(x) is the great-circle distance (the shortest possible distance on a sphere) to the target.



- The algorithm is searching for a path between Washington, D.C. and Los Angeles.



- Illustration of A* search for finding path from a start node to a goal node in a robot motion planning problem.
- The empty circles represent the nodes in the *open set*, i.e., those that remain to be explored, and the filled ones are in the closed set.
- Color on each closed node indicates the distance from the start: the greener, the farther.
- One can first see the A* moving in a straight line in the direction of the goal, then when hitting the obstacle, it explores alternative routes through the nodes from the open set.

Game engines as an industry

- Just as gaming is an industry, so are the engines they are built off. The major game engines come at varying prices, whether it be in the form of subscription fees or license payments.
- Unreal Engine 4, one of the major game engines and used to create several notable games such as *Fortnite*, *PlayerUnknown's Battlegrounds*, and *Life Is Strange 2*, adopted a free-to-use structure with a royalty on all game sales using this engine.
- Another game engine currently bringing in a notable income is the Unity engine, utilizing a similar pay module to the Unreal Engine. This engine is behind games such as *Rust*, *Subnautica*, and *Life Is Strange: Before the Storm*.

Game middleware

- The four most widely used middleware packages that provide subsystems of functionality include RAD Game Tools' Bink, Firelight FMOD, Havok, and Scaleform GFx.
- RAD Game Tools develops Bink for basic video rendering, along with Miles audio, and Granny 3D rendering.
- Firelight FMOD is a low cost robust audio library and toolset. Havok provides a robust physics simulation system, along with a suite of animation and behavior applications.
- *Scaleform* provides GFx for high performance Flash UI and high-quality video playback, and an Input Method Editor (IME) add-on for in-game Asian chat support.

- Other middleware is used for performance optimisation - for example 'Simplygon' helps to optimise and generate level of detail meshes, and 'Umbra' adds occlusion culling optimisations to 3d graphics.
- Some middleware contains full source code, others just provide an API reference for a compiled binary library. Some middleware programs can be licensed either way, usually for a higher fee for full source code.

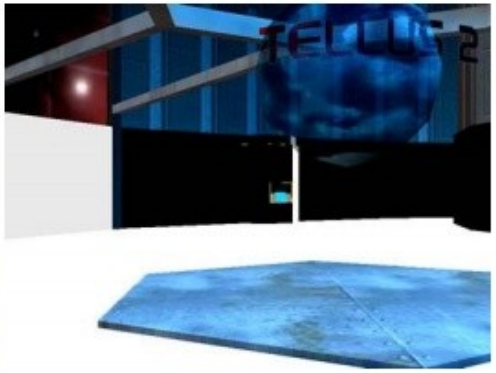
3 Dimensional game development

- Developing a 3D game demands in itself technical innovation in many areas like 3D graphics and other media optimization, game physics, game AI and so on.
- However, in this example will focus on the sound interface and how it relates to the graphic interface, since this combination is the most original aspect of Terraformers.
- Terraformers is the result of three years of practical research in developing a real-time 3D graphic game accessible for blind and low vision gamers as well as full sighted gamers.

- Direct orientation: using the numeric keyboard the gamer can orient the avatar directly in 8 directions (north, northeast...)
- High Contrast Mode: With this feature turned on, you get parts of the 3D graphics rendered in black and white toon-style which enhance contrast on important objects for low vision gamers.



3D scene rendered in normal contrast for sighted gamers



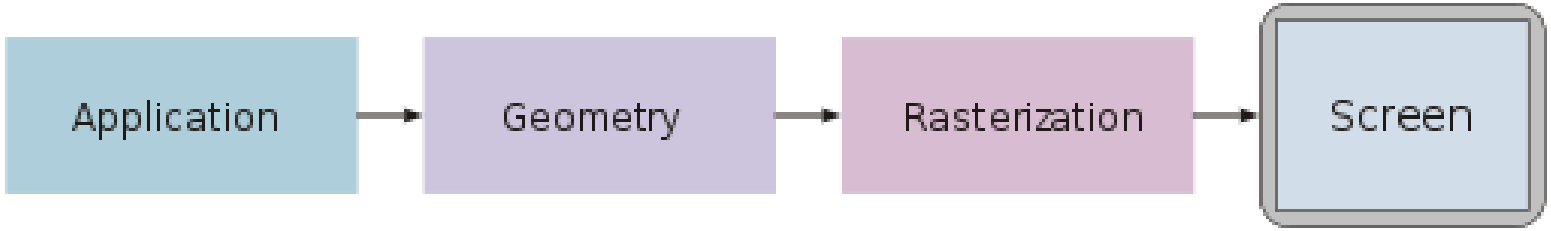
3D scene with important parts rendered in high contrast, for low vision gamers

- 3D Graphics Mode: The 3D graphic rendering can be completely turned off for blind gamers who don’t have 3D graphic hardware installed. There is also a possibility to set the 3D engine to not use 3D graphic hardware, i.e. it the use the regular CPU only which is a way to workaround malfunctioning graphic drivers in Windows (which some blind gamers don’t want to bother about).
- Backpack: objects are accessed using a voiced menu system with hierarchies. Sound effects can also be attached to voices dynamically.
- Game objects: all game objects have voiced feedback and 3D sound icons (and 3D graphics!)
- Environments: 3D graphics, ambient sounds, footstep sounds on different ground materials and voiced descriptions of visual as well as other sensory input contribute to environmental feedback.

- All the accessibility features for accessing the 3D world was considered easy to use (Sonar, GPS, Compass)
- The Backpack was easy to use.
- Since both the backpack and the GPS use a voiced menu system which also is used by other in game objects (like Communication devices) and also the game shell start menu, they are probably also easy to use.

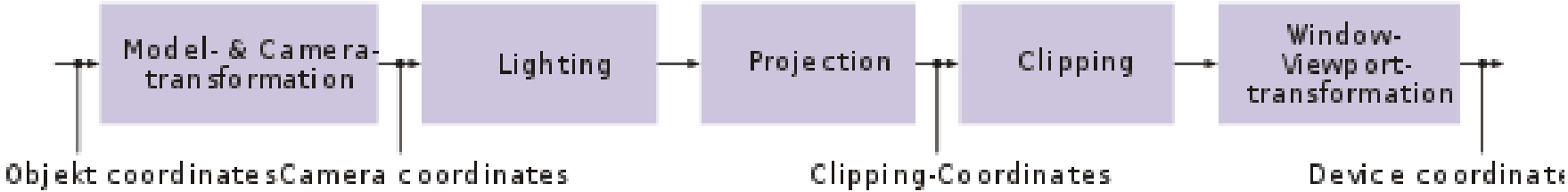
3D game pipeline

- The 3D pipeline usually refers to the most common form of computer 3D rendering called 3D polygon rendering, distinct from raytracing and raycasting.
- In raycasting, a ray originates at the point where the camera resides, and if that ray hits a surface, the color and lighting of the point on the surface where the ray hit is calculated.
- A graphics pipeline can be divided into three main parts: Application, Geometry and Rasterization

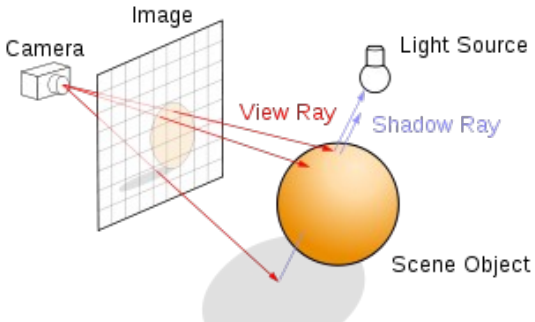


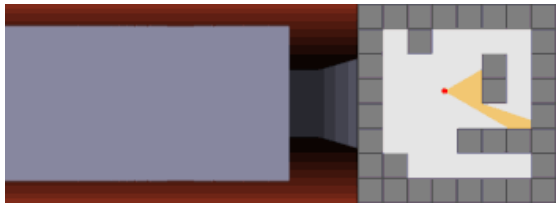
- The **application** step is executed by the software on the main processor (CPU).
- In the application step, changes are made to the scene as required, for example, by user interaction by means of input devices or during an animation.
- In a modern **Game Engine** such as **Unity**, the programmer deals almost exclusively with the application step, and uses a high-level language such as C#.
- The new scene with all its primitives, usually triangles, lines and points, is then passed on to the next step in the pipeline.

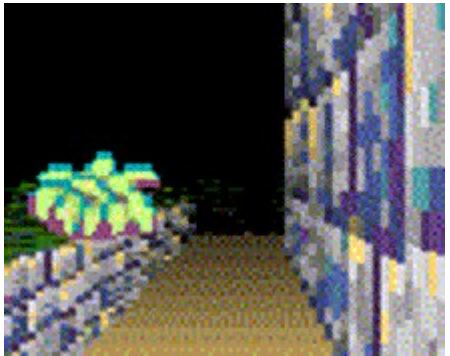
- The **geometry** step (with **Geometry pipeline**), which is responsible for the majority of the operations with polygons and their vertices (with Vertex pipeline), can be divided into the following five tasks.
- It depends on the particular implementation of how these tasks are organized as actual parallel pipeline steps.
- The **rasterisation** step is the final step before the fragment shader pipeline that all primitives are rasterized



- The ray-tracing algorithm builds an image by extending rays into a scene and bouncing them off surfaces and towards sources of light to approximate the color value of pixels in the first figure.
- A simple animation of ray casting in all directions in the second figure.
- Game demo using ray casting, making use of advanced techniques to render floor at multiple height levels. Camera shearing creates an effect of looking up or down in the third figure.







Cloud gaming

- Cloud gaming, sometimes called gaming on demand or gaming-as-a-service, is a type of online gaming that runs video games on remote servers and streams them directly to a user's device, or more colloquially, playing a game remotely from a cloud.
- It contrasts with traditional means of gaming, wherein a game runs locally on a user's video game console, personal computer, or mobile device.

GPU resource sharing

- A proposed method to improve game streaming's scalability is adaptive graphics processing unit (GPU) resource scheduling.
- Most cloud gaming providers are using dedicated GPUs to each person playing a game.
- This leads to the best performance but can waste resources.
- With better GPU resource scheduling algorithms, if the game does not fully utilize that GPU it can be used to help run someone else’s game simultaneously.
- In the past, “GPU virtualization was not used due to the inferior performance of the resource scheduling algorithm”.
- However new resource management algorithms have been developed that can allow up to 90% of the GPUs original power to be utilized even while being split among many users.

Video game console

- A video game console is an electronic or computer device that outputs a video signal or visual image to display a video game that one or more people can play through some type of game controller.
- These may be home consoles which are generally placed in a permanent location connected to a television or other display device and controlled with a separate game controller, or handheld consoles that include their own display unit and controller functions built into the unit and can be played anywhere.
- Hybrid consoles combine elements of both home and handheld consoles.
- Components are
  - Console unit
  - Controllers
  - Game media
  - External storage
  - Console add-ons