Doom Wiki

3,609
PAGES

ADD NEW PAGE

Doom rendering engine

The **Doom rendering engine** is the core of the game engine that powers Doom and its sequels, and that is used as a base to power other games by id Software licensees, notably Heretic, Hexen, and Strife. It was created by John Carmack, with auxiliary functions written by John Romero, Dave Taylor, and Paul Radek.[1] Originally developed on NeXT computers, it was ported to DOS for Doom's initial release, and later ported to several other operating systems and game consoles.

The source code for the Linux version of the Doom games was released to the public in 1997 under a license that granted rights to non-commercial use, and re-released under the GNU General Public License in 1999. As a result, dozens of user-developed source ports have been created which allow Doom to run on previously unsupported operating systems, often fix bugs (including the static limits noted below), and sometimes radically expand the engine's functionality with new features.

It is not a true "3D" engine (as it is not possible to look up and down properly, and one sector cannot be placed above or beneath another), but is however a fairly elegant system which allows pseudo-3D rendering. When first published, Doom was revolutionary and almost unique in its ability to provide a fast texture-mapped 3D environment on contemporary hardware — late-model 386 and early 486 PCs, without specialised 3D graphics hardware, running at clock speeds of around 25-33MHz.

Despite the simplicity and speed of the renderer, it has limitations. The base renderer relies on 16.16 fixed point numbers (whole numbers between -32,768 and 32,767 with fractions limited to multiples of 1/65,536). Due to such limitations, accuracy in small units is lost as the limited precision hinders accuracy especially when multiplying and dividing. High resolutions cause more graphical glitches especially above the 5,000 pixel resolution range, some glitches appear as field of view distortions along with floors and ceilings extending towards the horizon.

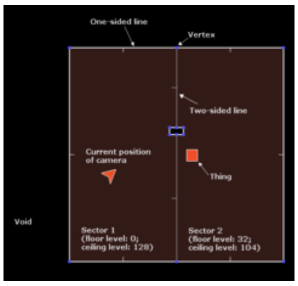
Contents [show]

Level structure

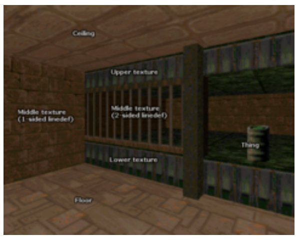
The following is only an overview of the basic structure of a Doom engine level. Most of the data structures listed here carry extra properties, such as texture offsets, or flags for restricting player or monster movement.

Viewed from the top down, all levels are actually two-dimensional, demonstrating one of the key limitations of the engine: it is not possible to have "rooms above rooms". This limitation, however, has a silver lining: a map mode can be easily displayed which represents the walls and the player's position, as shown schematically in the first image to the right (in contrast, the map mode of Doom's close contemporary Descent — which used an unfettered 3D engine [2] — was extremely hard to interpret).

A schematic diagram demonstrating how Doom represents levels internally


One-sided line
Vertex
Two-sided line
Current position of camera
Thing
Void
Sector 1 (floor level: 0; ceiling level: 128)
Sector 2 (floor level: 32; ceiling level: 104)

Map view in editor


Ceiling
Upper texture
Middle texture (1-sided linedef)
Middle texture (2-sided linedef)
Thing
Lower texture
Floor

In-game view

Basic objects

The base unit is the vertex, which signifies a single 2D point. In the diagram to the right, each small blue square is a vertex. Vertices (or "vertexes" as they are referred to internally) are then joined to form lines, known as linedefs. Each linedef can have either one or two sides, which are known as sidedefs. Sidedefs are then grouped together to form polygons; these are called sectors.

Sectors

Sectors represent particular 2D areas of the level. Each sector has a number of required properties: a floor height, a ceiling height, a light level, a floor texture, and a ceiling texture. To put two different light levels in the same room, for example, a new sector must be created for the second area. A sector must be completely surrounded by sidedefs; therefore, one-sided linedefs represent solid walls, while two-sided linedefs represent boundary lines between sectors.

Sidedefs

Sidedefs are used to store wall textures, which are completely independent of floor and ceiling textures. Each sidedef can have up to three textures; these are called the middle (or "normal"), upper, and lower textures. In one-sided linedefs, only the middle texture is used for the texture on the wall. In two-sided linedefs, the situation is more complex. The upper and lower textures are used to fill the gaps where adjacent sectors have different floor and ceiling heights: lower textures are used for stairs, for instance. Most sidedefs will not have a middle texture, although some do; this is used to make textures "hang" in mid-air. (For example, when a transparent bar texture is seen forming a cage, a middle texture has been used on a two-sided linedef.) An upper or lower texture on a one-sided linedef, or on a two-sided linedef whose adjoining sectors always have the same floor and ceiling heights, is highly unusual and may even cause rendering problems if not handled carefully by the level designer(s).

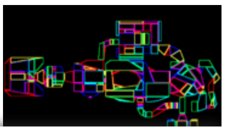
Things

Finally, there is a list of essentially non-architectural objects in the level, called things. These are used to place players, monsters, powerups, free-standing decorations, obstacles, etc. Each thing is given a 2D coordinate, as with the vertices. Things are then automatically placed on the floor or the ceiling as appropriate to their general type.

Node building

The Doom engine makes use of a system known as binary space partitioning (BSP). A tool, known as a node builder, must be used to generate the BSP data for a level before it can be played. Depending on the size and complexity of the level, this process can take quite some time.

BSP divides the level up into a binary tree: each location in the tree is a node which represents a particular area of the level (with the root node representing the entire level). At each branch of the tree there is a dividing line which splits the area of the node into two subnodes. At the same time, the dividing line divides linedefs into line segments called segs.



At the leaves of the tree are convex polygons, where it is not useful to divide the level up any further. These convex polygons are referred to as subsectors (or SSECTORS), and are bound to a particular sector. Each subsector has a list of segs associated with it.

The BSP system is really a very clever way of sorting the subsectors into the right order for rendering. The algorithm is fairly simple:

FANDOM POWERED BY WIKIA Games Movies TV Wikis START A WIKI

Games

Movies

TV

Explore Wikis

Follow Us

Overview
About
Careers
Press
Contact
Wikia.org

Terms of Use
Privacy Policy
Global Sitemap
Local Sitemap

Community
Community Central
Support
Fan Contributor Program
WAM Score
Help

Can't find a community you love? Create your own and start something epic.

START A WIKI

Community Apps
Take your favorite fandoms with you and never miss a beat
Download on the App Store GET IT ON Google Play

Advertise
Media Kit
Contact

Transformers Summer Sci-Fi Who Won Comic-Con? View Our SDCC Awards!