


An open source project

# **The Terminology Of** **Game Development**

*For game developers, artists and designers*

Published on 

## 1 Preface

There were a lot of words that I was unfamiliar with, as I started to learn game development.

So I decided to create my own "dictionary" to use as a reference for learning and making this path easier.

Then I thought, why not make this open source so anyone out there, who has similar issues would benefit from having this?

Therefore I made the decision to make this "terminology" open for everyone.

## 2 Contributors

Here's a list of all people who helped me gather this terminology book:

**Aien Saidi**

*[aien.saidi@gmail.com](mailto:aien.saidi@gmail.com)*

github: @aientech

twitter: @i\_in\_dev



## A

**Actor** • An Actor is any object that can be placed into a level (*See Level*). Actors are a generic Class that support 3D transformations such as translation, rotation, and scale. Actors

can be created (spawned) and destroyed through gameplay code (C++ or Blueprints (*See Blueprint*)). In C++, AActor is the base class of all Actors.

## B

**Blueprint** • Blueprints is the visual scripting system inside Unreal Engine 4 and is a fast way to start prototyping (*See Prototype*) your game. Instead

of having to write code line by line, you do everything visually: drag and drop nodes, set their properties in a UI, and drag wires to connect.

## D

**Directional Light** • The Directional Light simulates light that is being emitted from a source that is infinitely far away. This means that all

shadows cast by this light will be parallel, making this the ideal choice for simulating sunlight.

## E

**Edge** • A connection between two

vertices (*See Vertex*).

## F

**Face** • A closed set of edges (*See Edge*), in which a triangle face has

three edges, and a quad face has four edges.

## L

**Level** • When playing a video game, every object that you see or interact with resides in what is known as a Level. In Unreal Engine 4 terms, a Level is made up of a collection of Static Meshes (*See Mesh*), Volumes (*See Volume*), Lights (*See Light*), Blueprints (*See Blueprint*) and more all working together to bring the desired experience to the player. Levels in UE4 can range in size from massive terrain-based worlds to very small levels that contain a few Actors

(*See Actor*).

**Light** • Unreal Engine 4 has four light types:

- Directional (*See Directional Light*)
- Point (*See Point light*)
- Spot (*See Spot light*)
- Sky (*See Sky light*)
- Rect (*See Rect light*)

Directional lights are primarily used as your primary outdoor light or any light that needs to appear as if it is casting light from extreme or near-infinite distances. Point lights are your classic "light bulb" like light, emitting light in all directions from a single point. Spot lights emit light from a single point, but have their

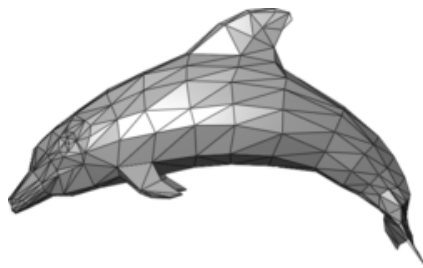
light limited by a set of cones. Sky lights capture the background of your scene and apply it as lighting to your level's meshes.

**Lighting Pass** • Emphasis is placed on improving lighting (*See Light*), adding PostProcess effects and updating materials (*See Material*).

## M

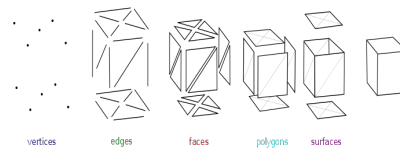
**Material** • A Material is an asset that can be applied to a mesh (*See Mesh*) to control the visual look of the scene. At a high level, it is probably easiest to think of a Material as the "paint" that is applied to an object.

**Mesh** • In 3D computer graphics and solid modeling, a polygon mesh is a collection of vertices, edges and faces that defines the shape of a polyhedral object. The faces usually consist of triangles (triangle mesh), quadrilaterals (quads), or other simple convex polygons (n-gons), since this simplifies rendering, but may also be more generally composed of concave polygons, or even polygons with holes.



Source: Wikipedia

A mesh is consisted of 5 components:



Source: Wikipedia

- Vertices (*See Vertex*)
- Edges (*See Edge*)
- Faces (*See Face*)
- polygons (*See Polygon*)
- Surfaces (*See Surface*)
- Materials (*See Material*)
- UV Coordinates (*See UV Coordinates*)

**Meshing Pass** • Replacing the basic meshes (*See Mesh*) or primitives from the prototype phase (*See Prototype Pass*) with near final asset and applying basic materials (*See Material*).

## P

**Point light** • Point Lights work much like a real-world light bulb, emitting light in all directions from the light bulb's tungsten filament. However, for the sake of performance, Point Lights are simplified down emitting light equally in all directions from just a single point in space.

**Polish Pass** • Additional effects, audio and volumes are added, and final assets and details are tweaked.

**Polygon** • A polygon is a coplanar set of faces (*See Face*). In systems that support multi-sided faces, polygons and faces are equivalent. However, most rendering hardware supports only 3- or 4-

sided faces, so polygons are represented as multiple faces. Mathematically a polygonal mesh may be considered an unstructured grid, or undirected graph, with additional properties of geometry, shape and topology.

**Prototype** • A first or preliminary version of a device or something from which other forms are developed.

**Prototype Pass** • The first pass (*See Workflow Pass*) when designing a level (*See Level*) is the prototype (*See Prototype*) pass. In this pass, a very general level (*See Level*) prototype is laid out using either Brushes or basic geometric Static Meshes such as cubes and spheres. Basic materials (*See Material*) can be ap-

plied to any Static Meshes, but this is more for being able to differentiate different objects or areas within your level (*See Level*), rather than for any aesthetic effect. Basic lighting is also added to the level (*See Level*) at this point.

The prototyping pass is meant to be a quick and simple process where the basic gameplay areas are roughed out. This allows for testing gameplay within the level (*See Level*), and observing things like the layout and relative sizing of areas within the level (*See Level*), as well as how that affects players' movement throughout the game. By the end of this pass, you can have a good sense of the playability and environment of your area.

## R

**Ray tracing** • Simply put, ray tracing is a technique that makes light in videogames behave like it does in real life. It works by simulating actual light rays, using an algorithm to trace the path that a beam of light would take in the physical world. Using this technique, game designers can make virtual rays of light appear to bounce off objects, cast realistic shadows, and

create lifelike reflections.

**Rect light** • The Rect Light emits light into the scene from a rectangular plane with a defined width and height. You can use them to simulate any kind of light sources that have rectangular areas, such as televisions or monitor screens, overhead lighting fixtures, or wall sconces.

## S

**Sky light** • The Sky Light captures the distant parts of your level (*See Level*) and applies that to the scene as a light. That means the sky's appearance and its lighting/reflections will match, even if your sky is coming from atmosphere, or layered clouds on top of a skybox, or distant mountains. You can also manually specify a cubemap to use.

**Spot light** • A Spot Light emits light from a single point in a cone shape. Users are given two cones to shape the light - the Inner Cone Angle and Outer Cone Angle. Within the Inner Cone Angle, the light achieves full brightness. As you go from the extent of the inner radius to the extents of the Outer Cone Angle, a falloff takes

place, creating a penumbra, or softening around the Spot Light's disc of illumination. The Radius of the light defines the length of the cones. More simply, this will work like a flash light or stage can light.

**Surface** • More often called smoothing groups, are useful, but not required to group smooth regions. Consider a cylinder with caps, such as a soda can. For smooth shading of the sides, all surface normals must point horizontally away from the center, while the normals of the caps must point straight up and down. Rendered as a single, Phong-shaded surface, the crease vertices (*See Vertex*) would have incorrect normals. Thus, some way of de-

termining where to cease smoothing is needed to group smooth parts of a mesh (*See Mesh*), just as polygons (*See Polygon*) group 3-sided faces (*See Face*). As an alternative to providing surfaces/smoothing groups, a mesh may contain other data for calculating the same data, such as a splitting angle (polygons with normals above this threshold are either automatically treated as separate smoothing groups or some technique such as splitting or cham-

fering is automatically applied to the edge between them). Additionally, very high-resolution meshes are less subject to issues that would require smoothing groups, as their polygons (*See Polygon*) are so small as to make the need irrelevant. Further, another alternative exists in the possibility of simply detaching the surfaces themselves from the rest of the mesh. Renderers do not attempt to smooth edges across noncontiguous polygons.

## U

**UV Coordinates** • Most mesh (*See Mesh*) formats also support some form of UV-coordinates which are a separate 2d representation of the mesh "unfolded" to show what portion of a 2-dimensional texture map to apply to different polygons

(*See Polygon*) of the mesh. It is also possible for meshes to contain another such vertex (*See Vertex*) attribute information such as colour, tangent vectors, weight maps to control animation, etc (sometimes also called channels).

## V

**Vertex** • A position (usually in 3D space) along with other information such as color, normal vector and texture coordinates.

**Volume** • Volumes are three-dimensional Actors (*See Actor*) used to alter the behavior of areas within levels. Volumes can be used for behaviors like:

- Causing damage to the player or

other Actors inside the Volume.

- Blocking certain Actors from entering the Volume, acting as a collision surface.
- Opening a door when an Actor enters the Volume.
- Changing the way a level calculates its lighting or visibility.

## W

**Workflow Pass** • A phase