Graphics Programming

Revision Notes

Ajil Pappachan

L5 GMD

Syllabus

UNIT-I:

Viewing Model - Camera Model - Orthographic Projection - Perspective Projection - Camera Control: Pitch - Yaw - Roll - 3D File format - Blending - Transparency - Stenciling - Depth and Stencil Buffer - Mouse Picking - Basic Lighting - Per-Pixel Lighting - Specular Lighting

UNIT-II:

Texturing: Creating Textures - Anisotropic filtering - Texture Tiling - Multiple Textures - Alpha Mapping - Normal Mapping - Reflection and Refraction with Cube maps - Projected Texture - Rendering to Texture - Bloom effect - Gamma correction - Multi sample Anti-Aliasing - Deferred Shading

UNIT-III:

Terrain: Flat Terrain - Height Map - Collision Detection - Light Types: Point Light - Spot Light - Directional Light - Global Illumination - Shadow - Shadow Mapping - Skybox - Atmospheric Light Scattering - Lens Flare - Occlusion Queries - Post-Processing: Bloom - Gaussian Blur - Physically Based Rendering - SSAO - Deferred Shading

UNIT-IV:

Introduction to Skeletal Animation - Skinned Mesh Animation using Matrices - Degree of Freedom - Hierarchy and Bone Weights - Animator - Blending Animation - Node Transformation - Water simulation: Frame Buffer Objects - Clipping Planes - Projective Texture Mapping - DuDv Maps - Fresnes Effect - Normal Maps .

UNIT-V:

Tessellation: Tessellation Stages - Tessellation Primitive Types - Tessellation Control - Tessellating Polygons - Geometry Shader: Shader Inputs and Outputs - Point Sprites - Vertex structure - Producing Primitives - Culling Geometry - Instancing - Layered Rendering - Compute Shader: Data Input and Output

Unit I

Viewing Model

By default, in OpenGL, the viewer is positioned on the z axis, it is like using a camera to take a shot. Imagine that your camera points to the origin of the Cartesian system. The up direction is parallel to the Oy axis and in the positive sense of Oy.

The view matrix in OpenGL controls the way we look at a scene. In this article we are going to use a view matrix that simulates a moving camera, usually named lookAt.

It is beyond the purpose of the present article to derive and present the way we create the view matrix, suffice to say that it is a 4x4 matrix, like the model matrix, and it is uniquely determined by 3 parameters:

The eye, or the position of the viewer;

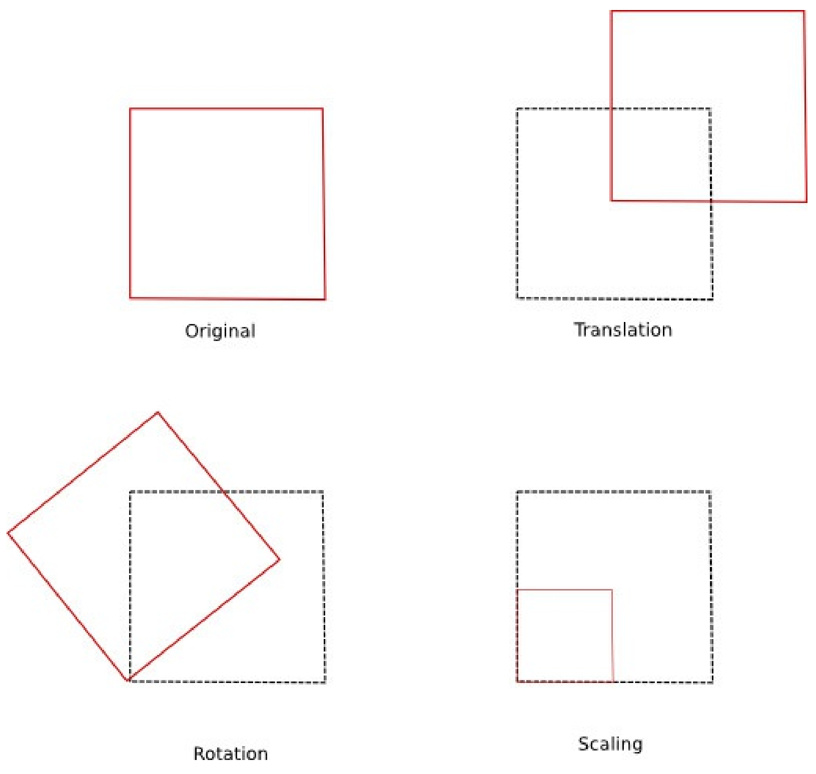
The center, or the point where we the camera aims;

The up, which defines the direction of the up for the viewer.

The defaults in OpenGL are: the eye at (0, 0, -1); the center at (0, 0, 0) and the up is given by the positive direction of the Oy axis (0, 1, 0).

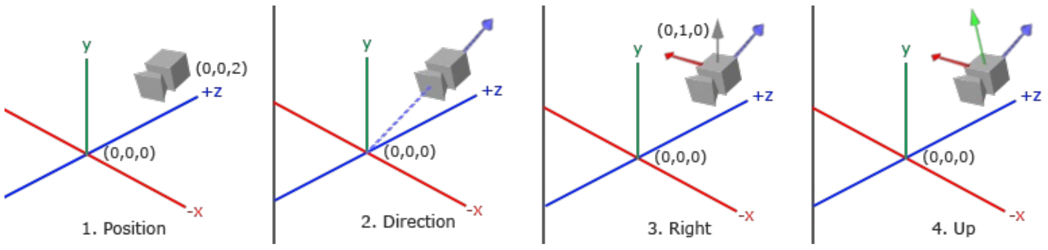
Suppose that we have a generic C++ function that given the eye, the center and the up will return a 4x4 view matrix for us.

The view matrix, V, multiplies the model matrix and, basically aligns the world (the objects from a scene) to the camera.



Camera Model

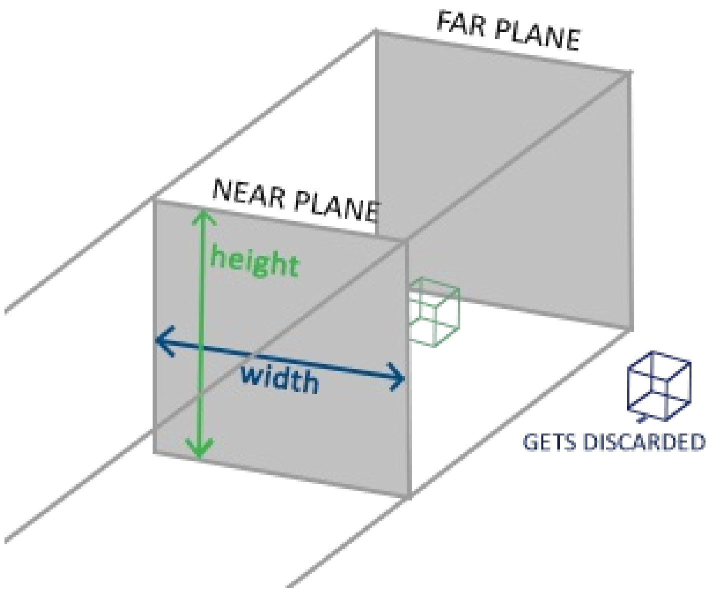
Camera model refers to all the vertex coordinates as seen from the camera's perspective as the origin of the scene: the view matrix transforms all the world coordinates into view coordinates that are relative to the camera's position and direction. To define a camera we need its position in world space, the direction it's looking at, a vector pointing to the right and a vector pointing upwards from the camera. A careful reader might notice that we're actually going to create a coordinate system with 3 perpendicular unit axes with the camera's position as the origin.



```
glm::vec3 cameraPos   = glm::vec3(0.0f, 0.0f,  3.0f);
glm::vec3 cameraFront = glm::vec3(0.0f, 0.0f, -1.0f);
glm::vec3 cameraUp    = glm::vec3(0.0f, 1.0f,  0.0f);

view = glm::lookAt(cameraPos, cameraPos + cameraFront, cameraUp);
```

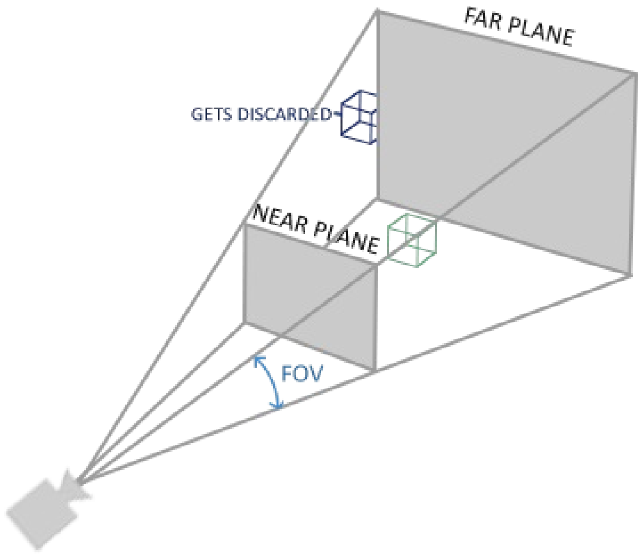
Orthographic Projection

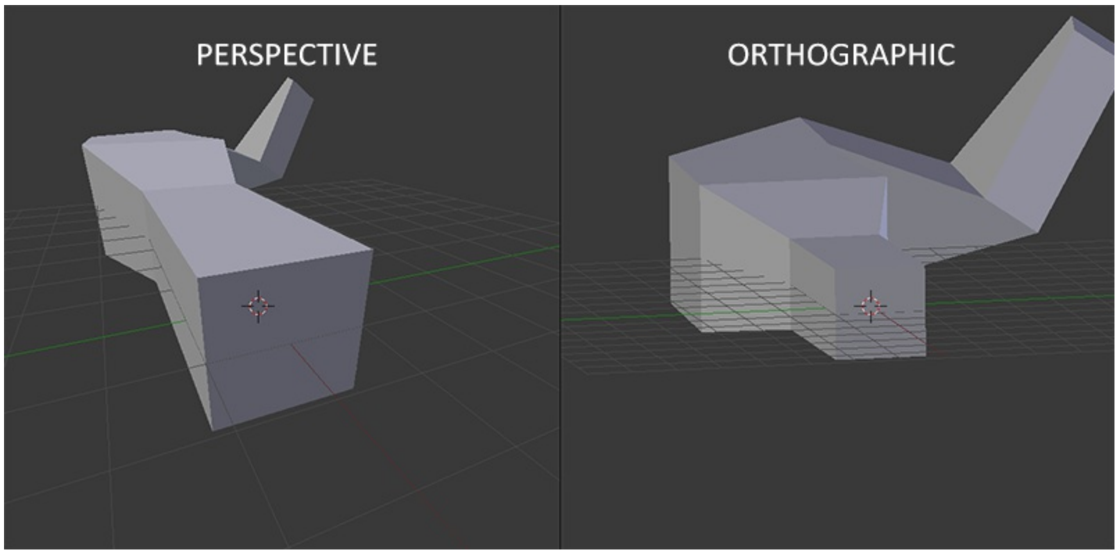
An orthographic projection matrix defines a cube-like frustum box that defines the clipping space where each vertex outside this box is clipped. When creating an orthographic projection matrix we specify the width, height and length of the visible frustum. All the coordinates that end up inside this frustum after transforming them to clip space with the orthographic projection matrix won't be clipped.



Perspective Projection

Perspective projection matrix maps a given frustum range to clip space, but also manipulates the w value of each vertex coordinate in such a way that the further away a vertex coordinate is from the viewer, the higher this w component becomes. Once the coordinates are transformed to clip space they are in the range -w to w (anything outside this range is clipped).

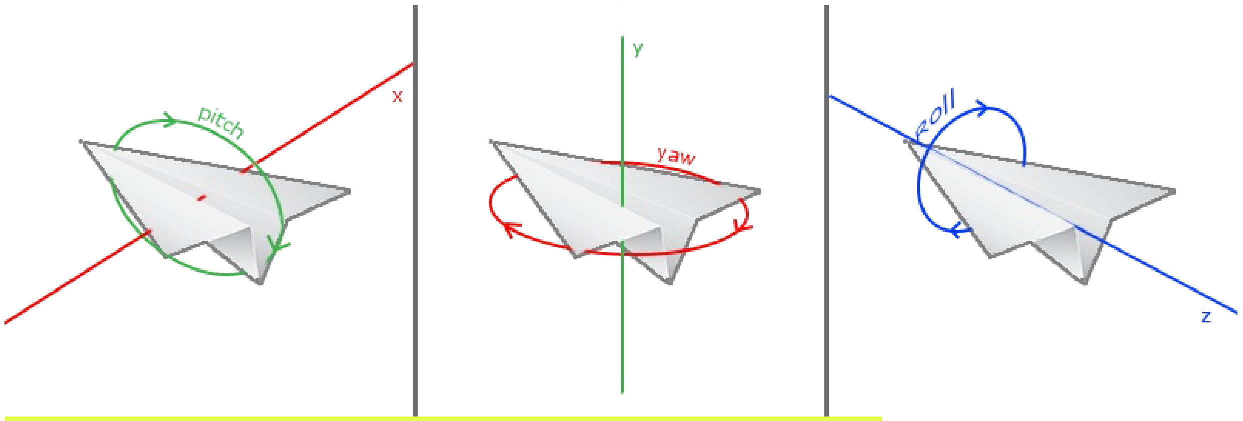

FAR PLANE
GETS DISCARDED
NEAR PLANE
FOV


PERSPECTIVE
ORTHOGRAPHIC

Camera Control:

Pitch, yaw and roll are the three dimensions of movement when an object moves through a medium.

Pitch: nose up or tail up.
Yaw: nose moves from side to side
Roll: a circular (clockwise or anticlockwise) movement of the body as it moves forward



```
direction.x = cos(glm::radians(pitch)) * cos(glm::radians(yaw));
direction.y = sin(glm::radians(pitch));
direction.z = cos(glm::radians(pitch)) * sin(glm::radians(yaw));
```

3D File format

there are dozens of different file formats where each exports the model data in its own unique way. Model formats like the Wavefront .obj only contains model data with minor material information like model colors and diffuse/specular maps, while model formats like the XML-based Collada file format are extremely extensive and contain models, lights, many types of materials, animation data, cameras, complete scene information and much more. The wavefront object format is generally considered to be an easy-toparse model format.

Blending

Blending in OpenGL is also commonly known as the technique to implement transparency within objects. Transparency is all about objects (or parts of them) not having a solid color, but having a combination of colors from the object itself and any other object behind it with varying intensity. A colored glass window is a transparent object; the glass has a color of its own, but the resulting color contains the colors of all the objects behind the glass as well.

Transparency

Transparent objects can be completely transparent (it lets all colors through) or partially transparent (it lets colors through, but also shows some of its own colors). The amount of transparency of an object is defined by its color's alpha value. The alpha color value is the 4th component of a color vector.

Full transparent window

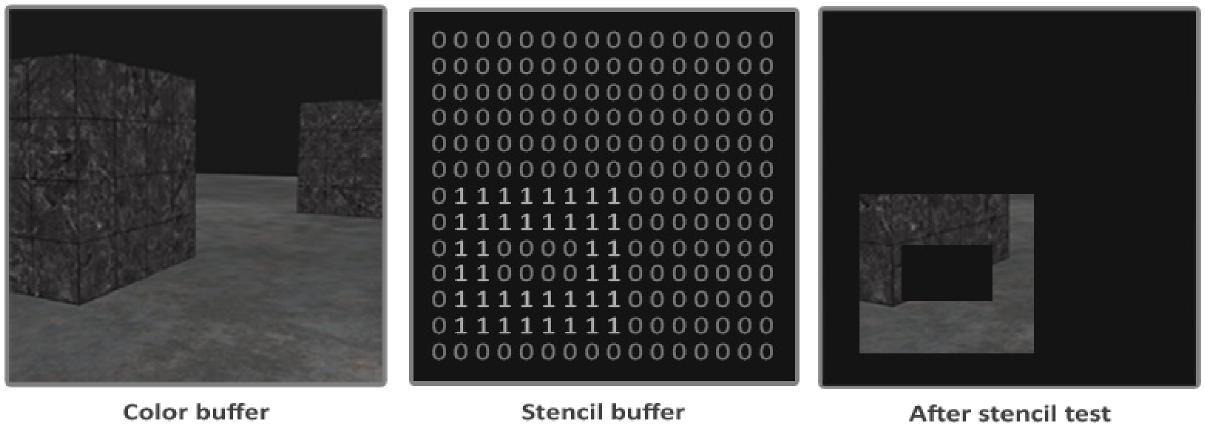
Partially transparent window

Stenciling

A stencil buffer (usually) contains 8 bits per stencil value that amounts to a total of 256 different stencil values per pixel/fragment. We can then set these stencil values to values of our liking and then we can discard or keep fragments whenever a particular fragment has a certain stencil value.

Depth and Stencil Buffer

Once the fragment shader has processed the fragment a so called stencil test is executed that, just like the depth test, has the possibility of discarding fragments. Then the remaining fragments get passed to the depth test that could possibly discard even more fragments. The stencil test is based on the content of yet another buffer called the stencil buffer.



Mouse Picking

Mouse picking is the most commonly used intuitive operation to interact with 3D scenes in a variety of 3D graphics applications. High performance for such operation is necessary in order to provide users with fast responses.

Basic Lighting

Lighting in OpenGL is based on approximations of reality using simplified models that are much easier to process and look relatively similar. These lighting models are based on the physics of light as we understand it. One of those models is called the Phong lighting model. The major building blocks of the Phong model consist of 3 components: ambient, diffuse and specular lighting.

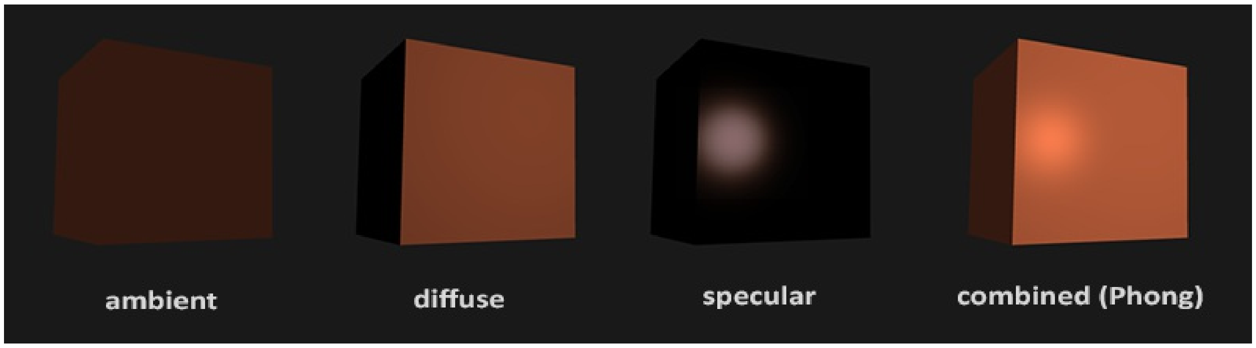
Per-Pixel Lighting

In computer graphics, per-pixel lighting refers to any technique for lighting an image or scene that calculates illumination for each pixel on a rendered image. This is in contrast to other popular methods of lighting such as vertex lighting, which calculates illumination at each vertex of a 3D model and then interpolates the resulting values over the model's faces to calculate the final per-pixel color values.

Per-pixel lighting is commonly used with techniques like normal mapping, bump mapping, specularity, and shadow volumes. Each of these techniques provides some additional data about the surface being lit or the scene and light sources that contributes to the final look and feel of the surface.

Specular Lighting

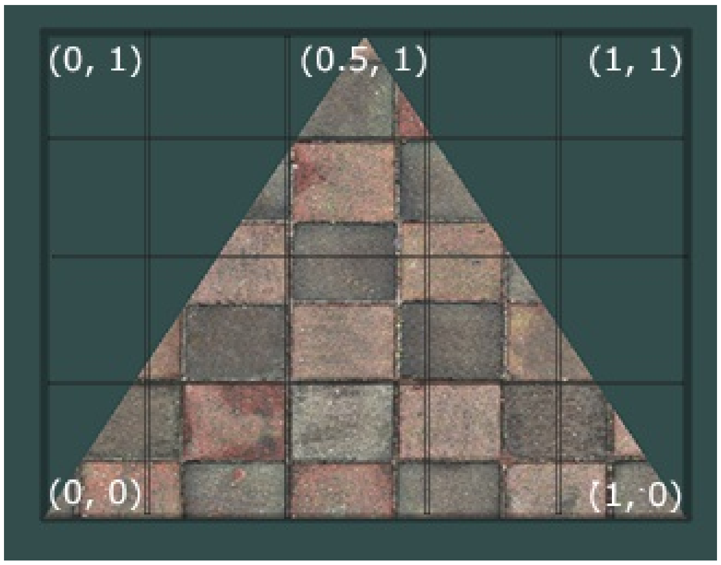
specular lighting is based on the light's direction vector and the object's normal vectors, but this time it is also based on the view direction e.g. from what direction the player is looking at the fragment. Specular lighting is based on the reflective properties of light. If we think of the object's surface as a mirror, the specular lighting is the strongest wherever we would see the light reflected on the surface.


ambient
diffuse
specular
combined (Phong)

Unit II - Texturing

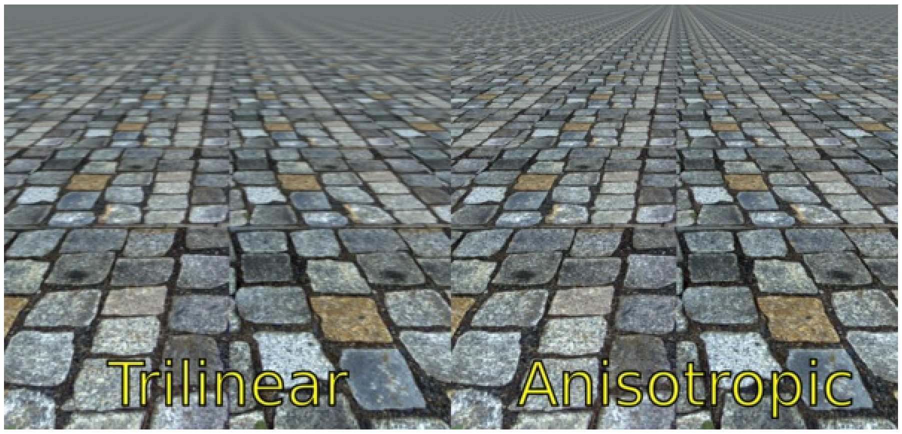
Creating Textures

A texture is a 2D image (even 1D and 3D textures exist) used to add detail to an object.



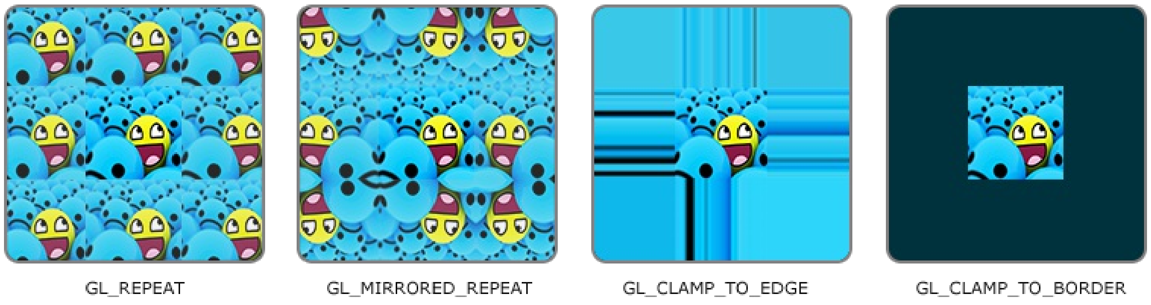
Anisotropic filtering

In 3D computer graphics, anisotropic filtering (abbreviated AF) is a method of enhancing the image quality of textures on surfaces of computer graphics that are at oblique viewing angles with respect to the camera where the projection of the texture (not the polygon or other primitive on which it is rendered) appears to be non-orthogonal (thus the origin of the word: "an" for not, "iso" for same, and "tropic" from tropism, relating to direction; anisotropic filtering does not filter the same in every direction).



Texture Tiling

A tiled texture looks best if its edges seamlessly match up with each other, top to bottom and side to side. Tiling is a common method of using the smallest texture possible to cover a large area, like a small texture of a couple bricks tiling across a large polygon, creating a big brick wall.

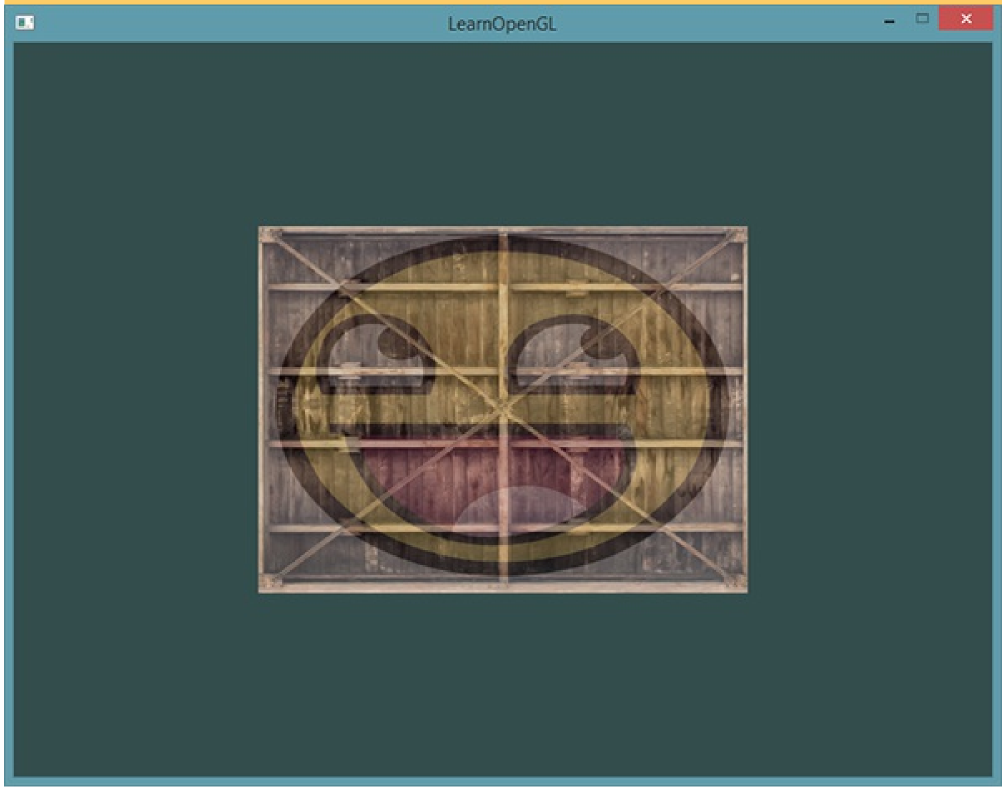


Multiple Textures

we can actually assign a location value to a texture sampler so we can set multiple textures at once in a fragment shader. This location of a texture is more commonly known as a texture unit. The main purpose of texture units is to allow us to use more than 1 texture in our shaders. By assigning texture units to the samplers, we can bind to multiple textures at once as long as we activate the corresponding texture unit first.

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture2);

glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

```
#version 330 core
...
uniform sampler2D texture1;
uniform sampler2D texture2;
void main()
{
    FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);
}
```



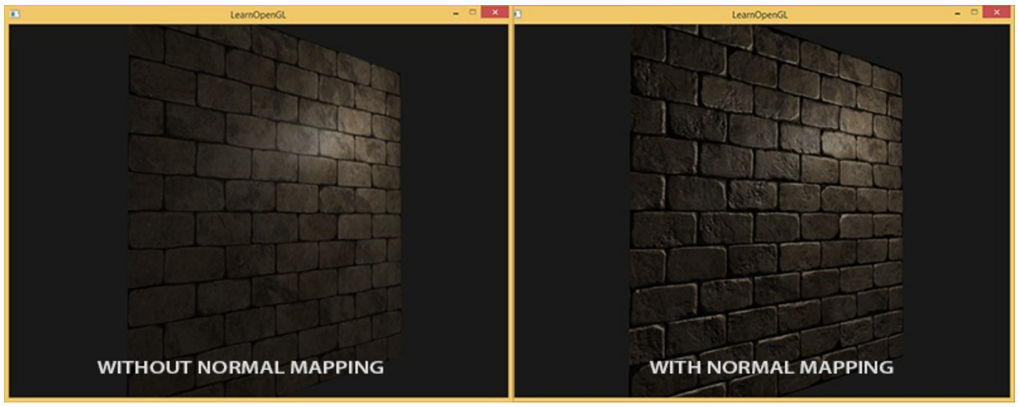
Alpha Mapping

Alpha mapping is a technique in 3D computer graphics involving the use of texture mapping to designate the amount of transparency/translucency of areas in a certain object.

Alpha mapping is used when the given object's transparency is not consistent: when the transparency amount is not the same for the entire object and/or when the object is not entirely transparent. If the object has the same level of transparency everywhere, one can either use a solid-color alpha texture or an integer value.

Normal Mapping

Normal mapping, or Dot3 bump mapping, is a technique used for faking the lighting of bumps and dents – an implementation of bump mapping. It is used to add details without using more polygons. A common use of this technique is to greatly enhance the appearance and details of a low polygon model by generating a normal map from a high polygon model or height map.

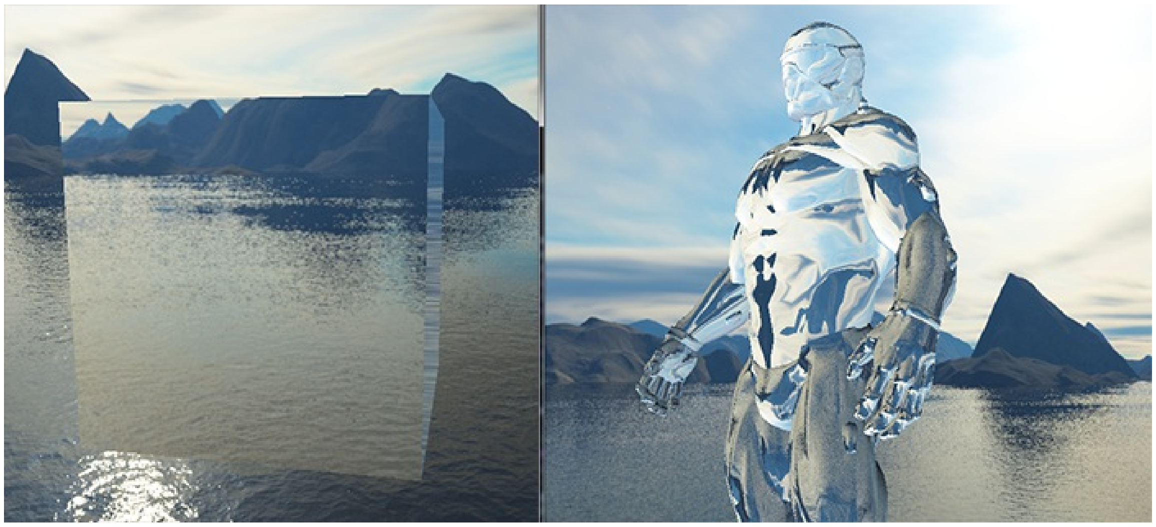


Reflection and Refraction with Cube maps

A cubemap is basically a texture that contains 6 individual 2D textures that each form one side of a cube: a textured cube.

Reflection is the property that an object (or part of an object) reflects its surrounding environment e.g. the object's colors are more or less equal to its environment based on the angle of the viewer. A mirror for example is a reflective object: it reflects its surroundings based on the viewer's angle. We calculate a reflection vector R' around the object's normal vector N' based on the view direction vector I'. We can calculate this reflection vector using GLSL's built-in reflect function. The resulting vector $\bar{R}$ is then used as a direction vector to index/sample the cubemap returning a color value of the environment. The resulting effect is that the object seems to reflect the skybox.

Refraction is the change in direction of light due to the change of the material the light flows through. Refraction is what we commonly see with water-like surfaces where the light doesn't enter straight through, but bends a little. Again, we have a view vector I', a normal vector N' and this time a resulting refraction vector R'. This resulting bended vector R' is then used to sample from the cubemap.



Projected Texture

Projective texture mapping is a method of texture mapping that allows a textured image to be projected onto a scene as if by a slide projector. Projective texture mapping is useful in a variety of lighting techniques and it is the starting point for shadow mapping.

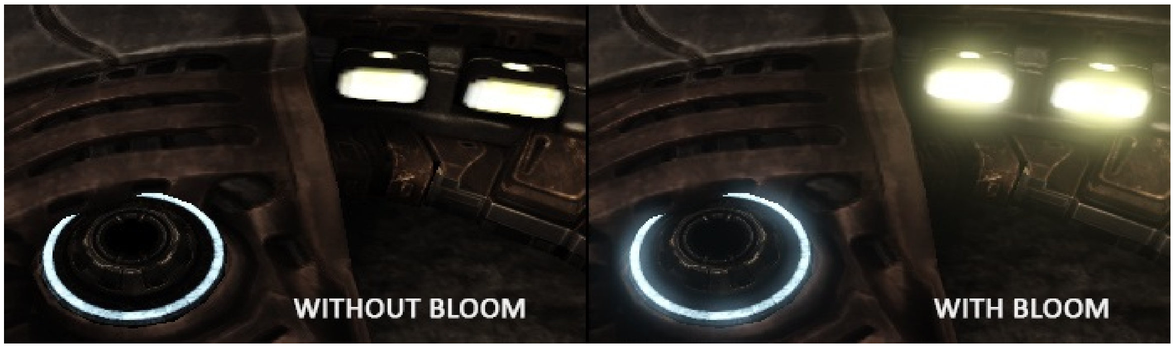
Rendering to Texture

Render-To-Texture is a handful method to create a variety of effects. The basic idea is that you render a scene just like you usually do, but this time in a texture that you can reuse later. Applications include in-game cameras, post-processing, and as many GFX as you can imagine.

Bloom Effect

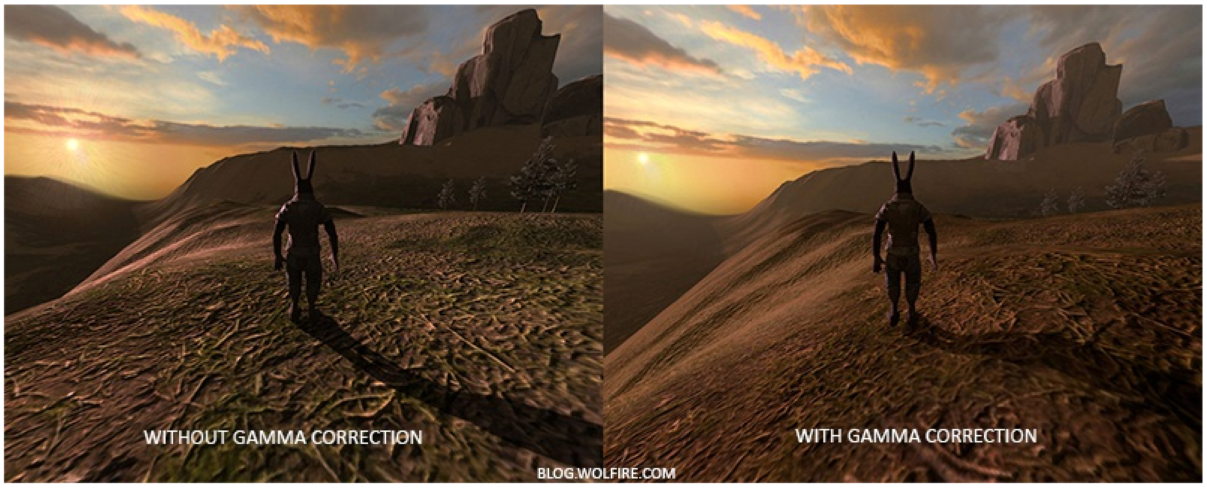
Bright light sources and brightly lit regions are often difficult to convey to the viewer as the intensity range of a monitor is limited. One way to distinguish bright light sources on a monitor is by making them glow, their light bleeds around the light source. This effectively gives the viewer the illusion these light sources or bright regions are intensily bright.

This light bleeding or glow effect is achieved with a post-processing effect called bloom. Bloom gives all brightly lit regions of a scene a glow-like effect.



Gamma Correction

The idea of gamma correction is to apply the inverse of the monitor's gamma to the final output color before displaying to the monitor.



Multi sample Anti-Aliasing

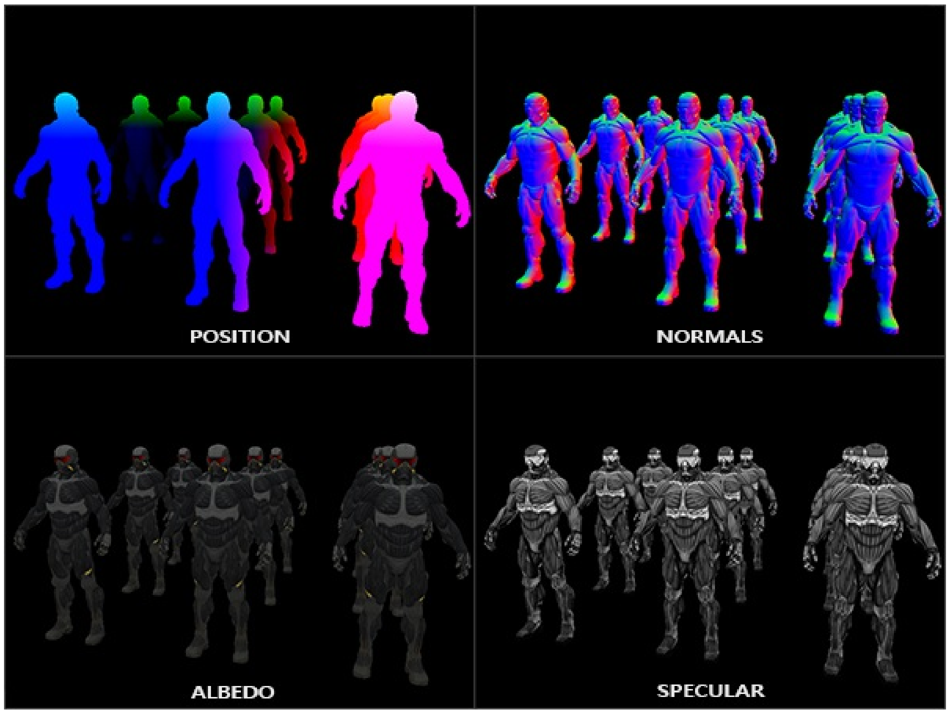
Initial implementations of full-scene anti-aliasing (FSAA) worked conceptually by simply rendering a scene at a higher resolution, and then downsampling to a lower-resolution output. Most modern GPUs are capable of this form of anti-aliasing, but it greatly taxes resources such as texture, bandwidth, and fillrate.

"multisampling" refers to a specific optimization of supersampling. The specification dictates that the renderer evaluate the fragment program once per pixel, and only "truly" supersample the depth and stencil values.

Deferred Shading

Deferred shading is based on the idea that we defer or postpone most of the heavy rendering (like lighting) to a later stage. Deferred shading consists of two passes: in the first pass called the geometry pass we render the scene once and retrieve all kinds of geometrical information from the objects that we store in a collection of textures called the G-buffer; think of position vectors, color vectors, normal vectors and/or specular values. The geometric information of a scene stored in the G-buffer is then later used for (more complex) lighting calculations.

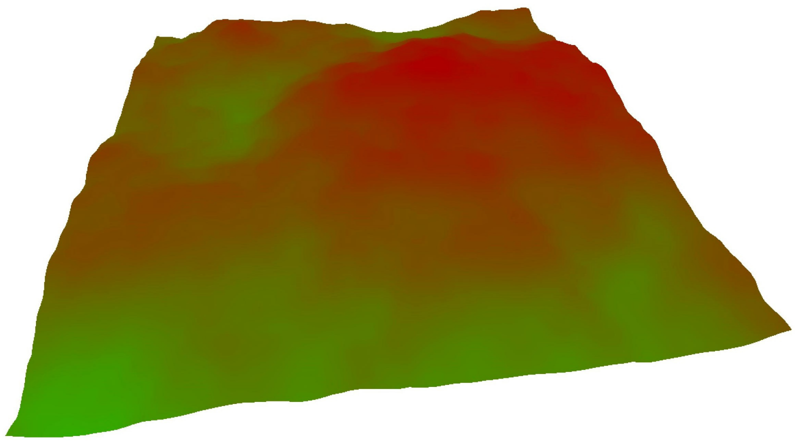
This gives us several new options to significantly optimize scenes with large numbers of lights, allowing us to render hundreds or even thousands of lights with an acceptable framerate.



Unit III - Terrain

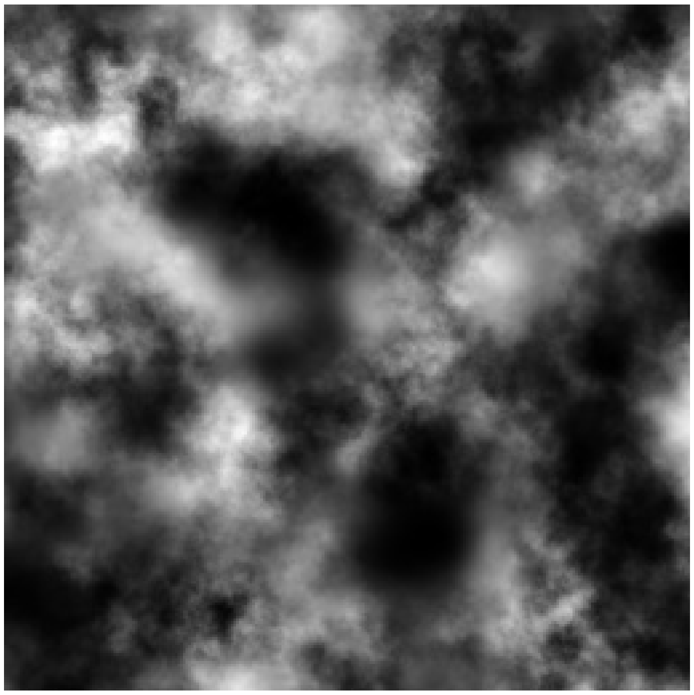
Flat Terrain

Terrain is a rectangular grid with varying Y-axis values (height) for each of it's polygons.



Height Map

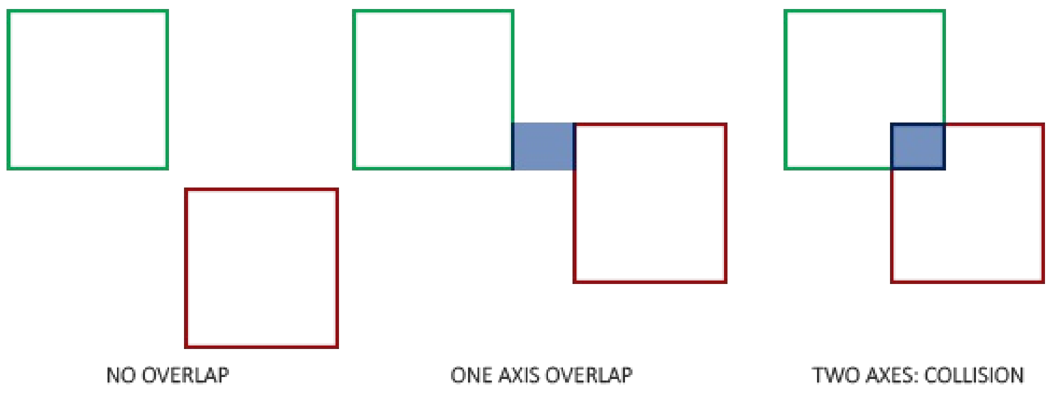
a heightmap or heightfield is a raster image used mainly as Discrete Global Grid in secondary elevation modeling. Each pixel store values, such as surface elevation data, for display in 3D computer graphics. A heightmap can be used in bump mapping to calculate where this 3D data would create shadow in a material, in displacement mapping to displace the actual geometric position of points over the textured surface, or for terrain where the heightmap is converted into a 3D mesh.



Collision Detection

it is a common practice to use more simple shapes (that usually have a nice mathematical definition) for collision detection that we overlay on top of the original object. We then check for collisions based on these simple shapes which makes the code easier and saves a lot of performance. Several examples of such collision shapes are circles, spheres, rectangles and boxes; these are a lot simpler to work with compared to meshes with hundreds of triangles.

```
GLboolean CheckCollision(GameObject &one, GameObject &two) // AABB - AABB
collision
{
    // Collision x-axis?
    bool collisionX = one.Position.x + one.Size.x >= two.Position.x &&
        two.Position.x + two.Size.x >= one.Position.x;
    // Collision y-axis?
    bool collisionY = one.Position.y + one.Size.y >= two.Position.y &&
        two.Position.y + two.Size.y >= one.Position.y;
    // Collision only if on both axes
    return collisionX && collisionY;
}
```

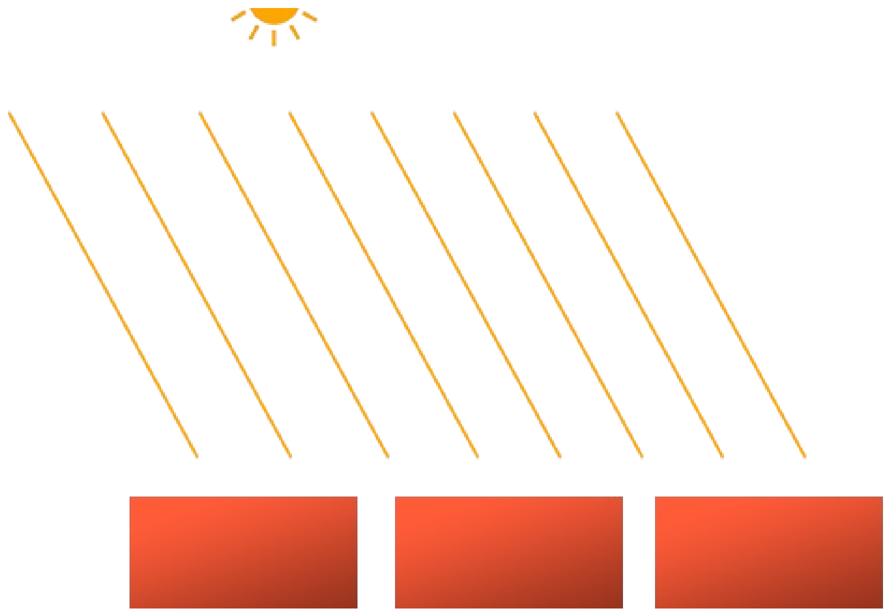


Light Types:

A light source that casts light upon objects is called a light caster.

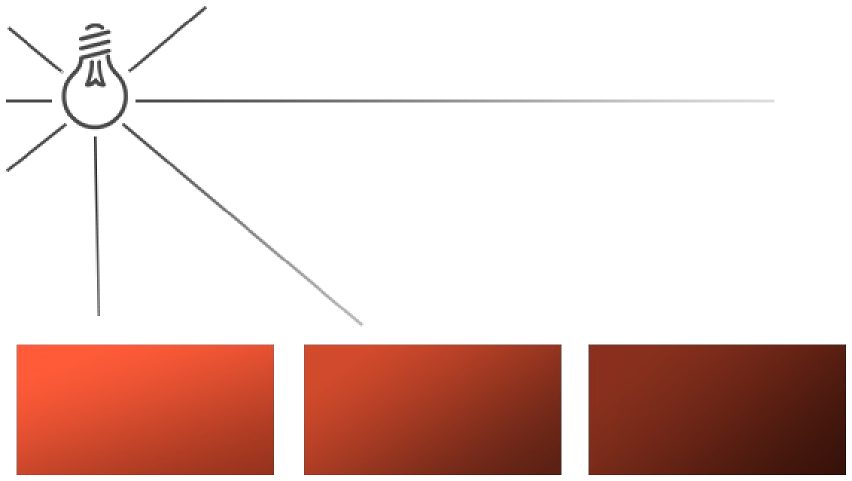
Directional Light

When a light source is far away the light rays coming from the light source are close to parallel to each other. It looks like all the light rays are coming from the same direction, regardless of where the object and/or the viewer is. When a light source is modeled to be infinitely far away it is called a directional light since all its light rays have the same direction; it is independent of the location of the light source.



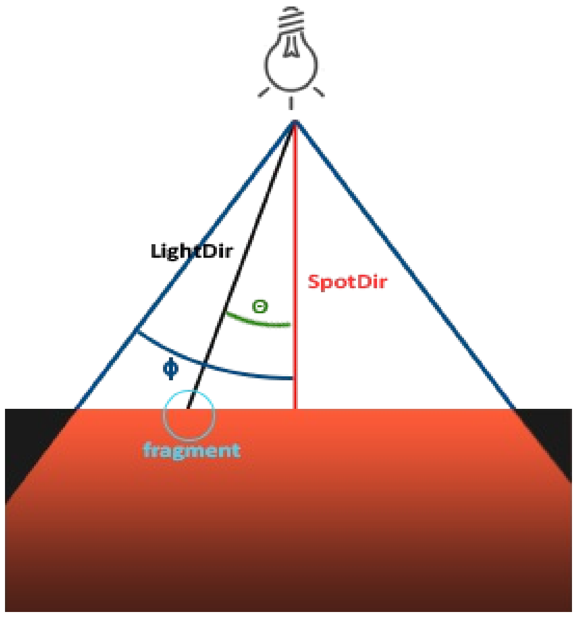
Point lights

Directional lights are great for global lights that illuminate the entire scene, but aside from a directional light we usually also want several point lights scattered throughout the scene. A point light is a light source with a given position somewhere in a world that illuminates in all directions where the light rays fade out over distance. Think of light bulbs and torches as light casters that act as a point light.



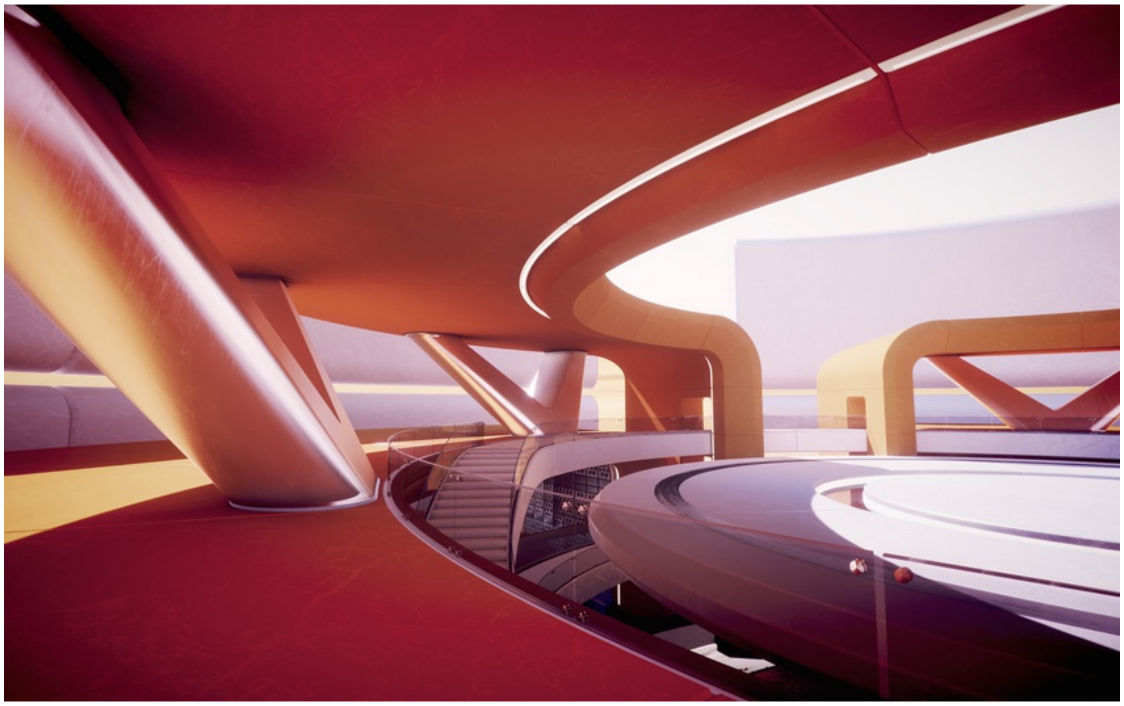
Spotlight

A spotlight is a light source that is located somewhere in the environment that, instead of shooting light rays in all directions, only shoots them in a specific direction. The result is that only the objects within a certain radius of the spotlight's direction are lit and everything else stays dark. A good example of a spotlight would be a street lamp or a flashlight.



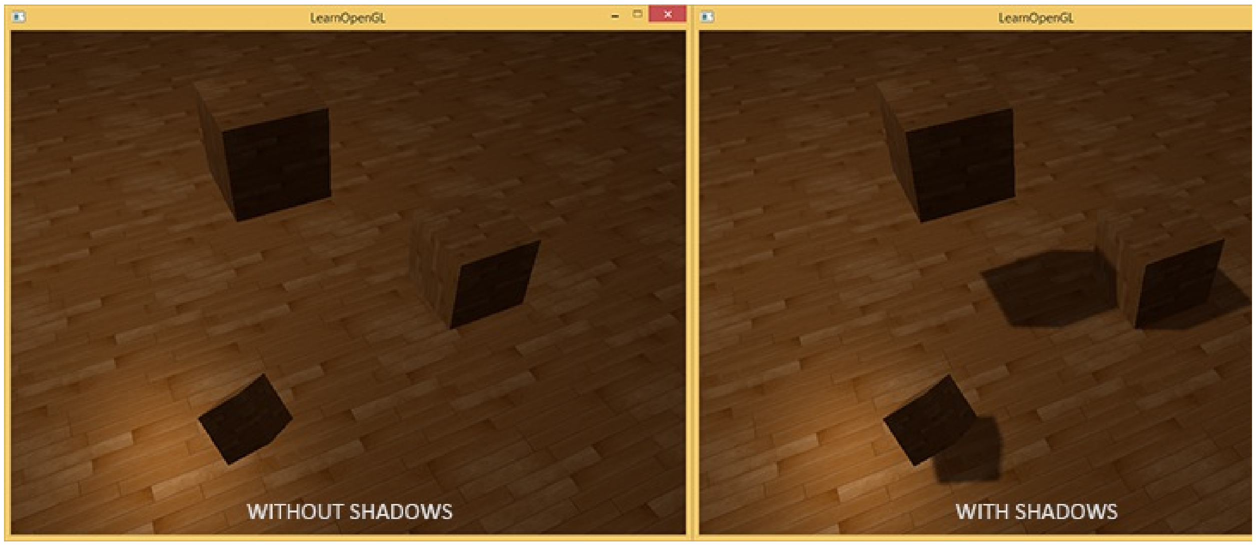
Global Illumination

Global Illumination (GI) is a system that models how light is bounced off of surfaces onto other surfaces (indirect light) rather than being limited to just the light that hits a surface directly from a light source (direct light). Modelling indirect lighting allows for effects that make the virtual world seem more realistic and connected, since objects affect each other's appearance. One classic example is 'color bleeding' where, for example, sunlight hitting a red sofa will cause red light to be bounced onto the wall behind it. Another is when sunlight hits the floor at the opening of a cave and bounces around inside so the inner parts of the cave are illuminated too.



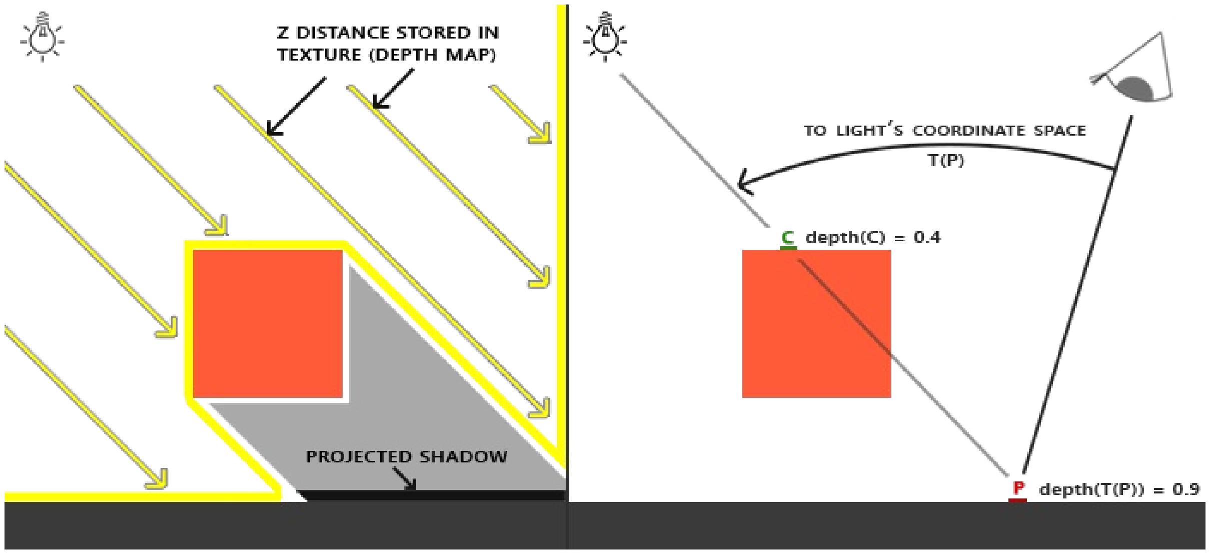
Shadow

Shadows are a result of the absence of light due to occlusion; when a light source's light rays do not hit an object because it gets occluded by some other object the object is in shadow. Shadows add a great deal of realism to a lighted scene and make it easier for a viewer to observe spatial relationships between objects.



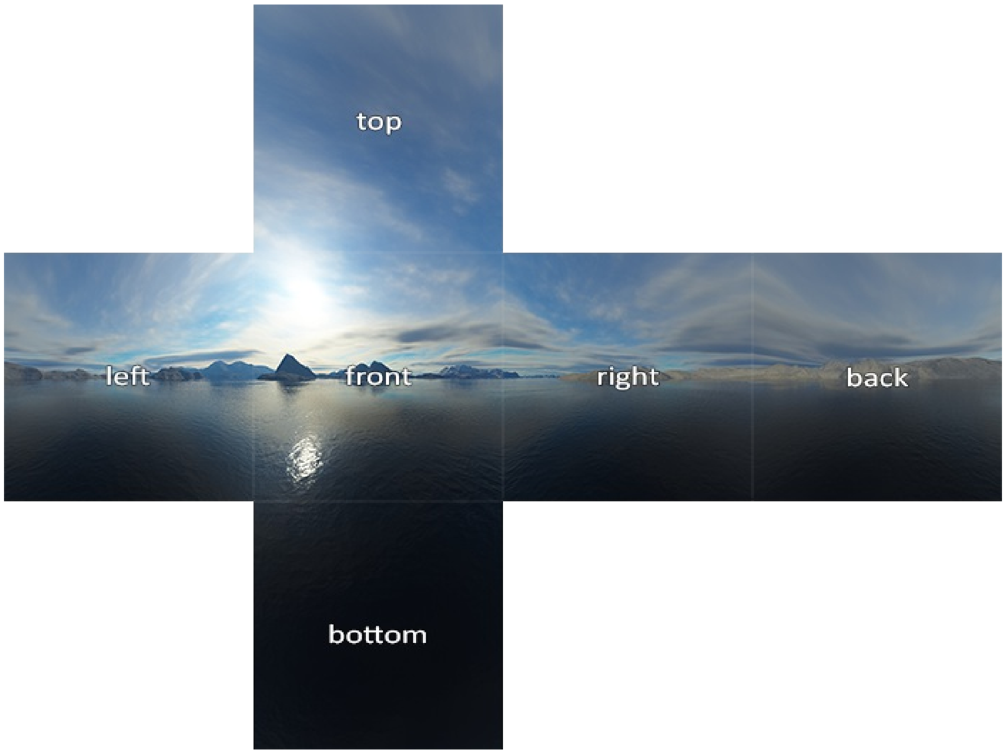
Shadow Mapping

Shadow mapping or shadowing projection is a process by which shadows are added to 3D computer graphics. Shadows are created by testing whether a pixel is visible from the light source, by comparing the pixel to a z-buffer or depth image of the light source's view, stored in the form of a texture.


Z DISTANCE STORED IN TEXTURE (DEPTH MAP)
PROJECTED SHADOW
TO LIGHT'S COORDINATE SPACE
T(P)
C depth(C) = 0.4
P depth(T(P)) = 0.9

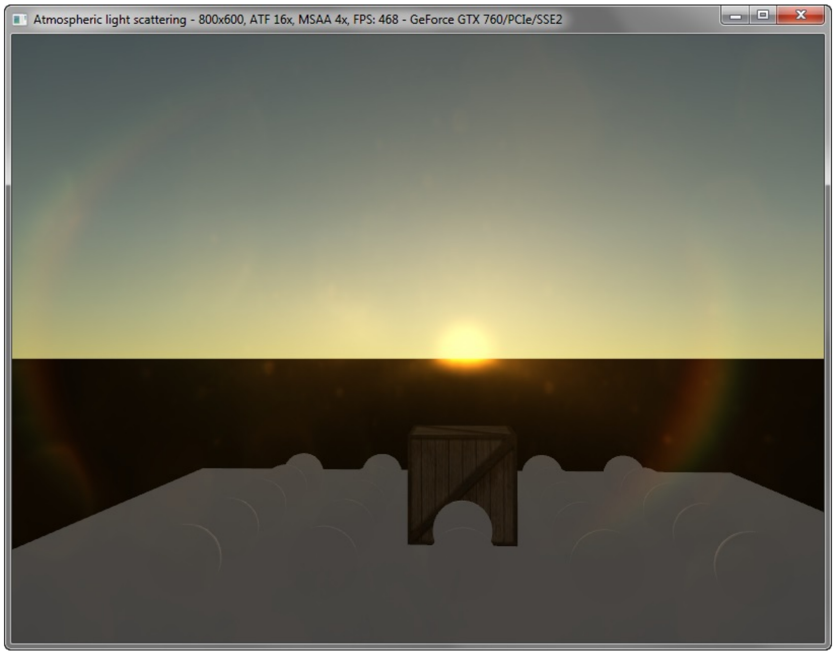
Skybox

A skybox is a (large) cube that encompasses the entire scene and contains 6 images of a surrounding environment, giving the player the illusion that the environment he's in is actually much larger than it actually is. Some examples of skyboxes used in videogames are images of mountains, of clouds or of a starry night sky.



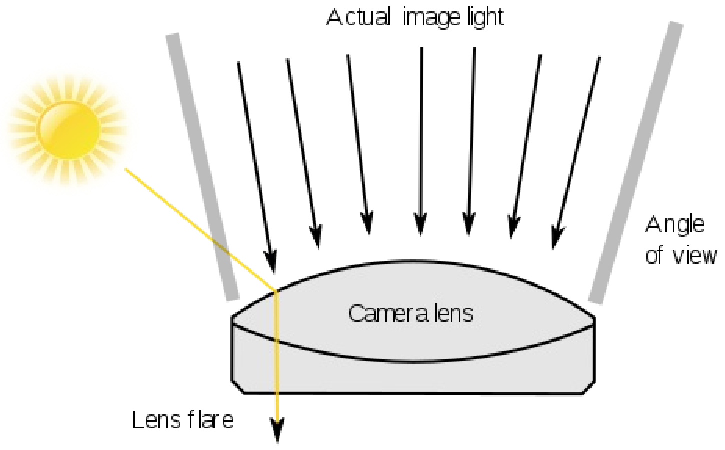
Atmospheric Light Scattering

In addition to being absorbed or transmitted, electromagnetic radiation can also be scattered by particles in the atmosphere. Scattering is the redirection of electromagnetic energy by suspended particles in the atmosphere.


Atmospheric light scattering - 800x600, ATF 16x, MSAA 4x, FPS: 468 - GeForce GTX 760/PCIe/SSE2

Lens Flare

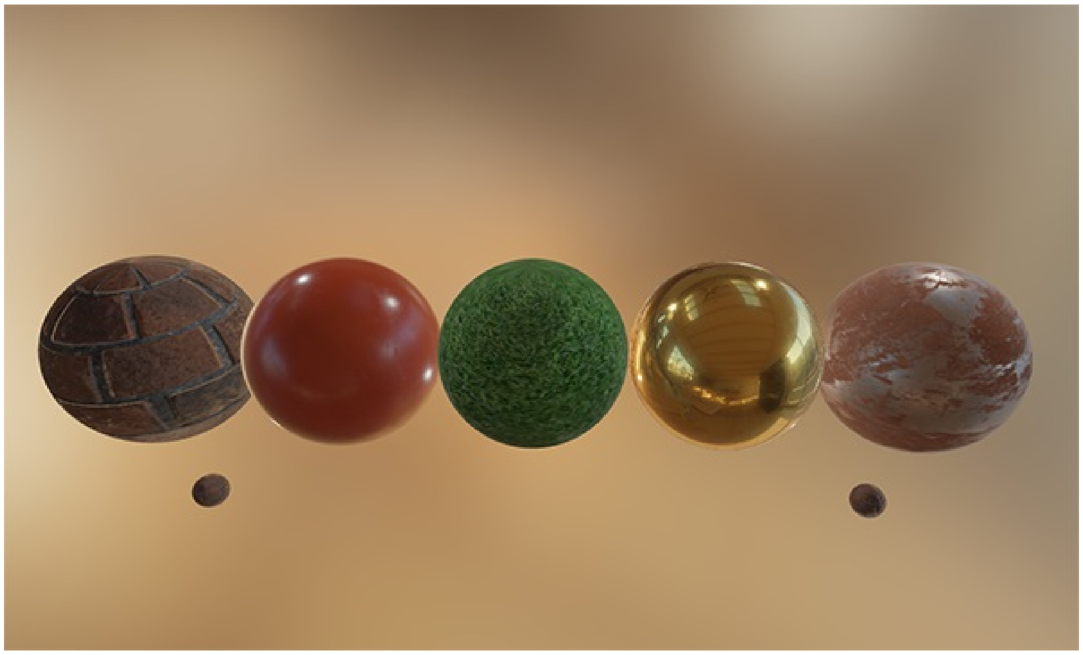
Lens flare refers to a phenomenon wherein light is scattered or flared in a lens system, often in response to a bright light, producing a sometimes undesirable artifact within the image. This happens through light scattered by the imaging mechanism itself, for example through internal reflection and scattering from material imperfections in the lens. Lenses with large numbers of elements such as zooms tend to exhibit greater lens flare, as they contain a relatively large number of interfaces at which internal scattering may occur. These mechanisms differ from the focused image generation mechanism, which depends on rays from the refraction of light from the subject itself.


Actual image light
Angle of view
Camera lens
Lens flare

Physically Based Rendering

Physically Based Rendering (PBR) is a collection of render techniques that are more or less based on the same underlying theory which more closely matches that of the physical world. As physically based rendering aims to mimic light in a physically plausible way it generally looks more realistic compared to our original lighting algorithms like Phong and Blinn-Phong. Not only does it look better, as it closely approximates actual physics, we (and especially the artists) can author surface materials based on physical parameters without having to resort to cheap hacks and tweaks to make the lighting look right. One of the bigger advantages of authoring materials based on physical parameters is that these materials will look correct regardless of lighting conditions; something that is not true in non-PBR pipelines.

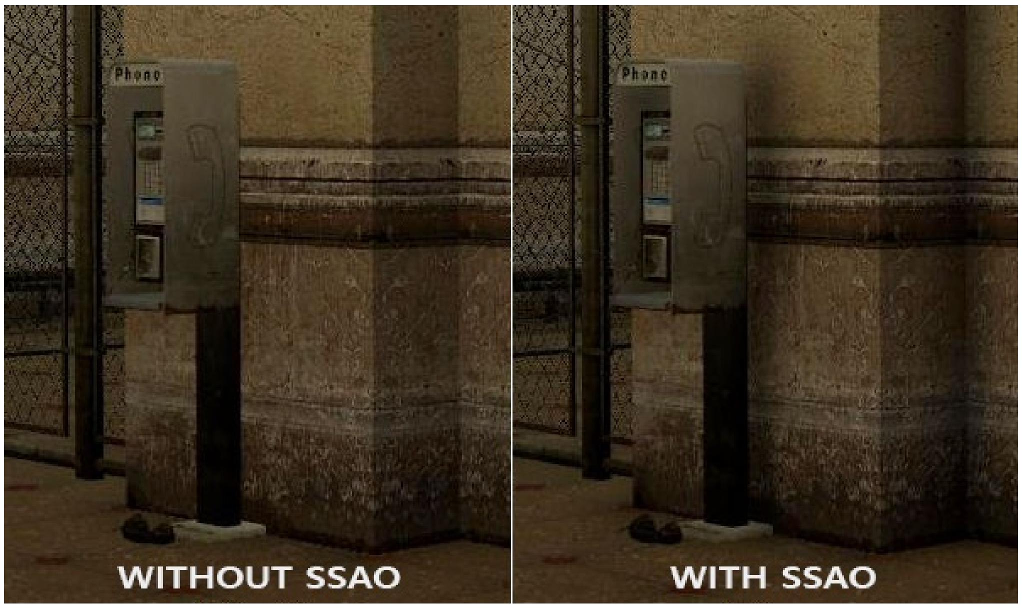
For a PBR lighting model to be considered physically based it has to satisfy the following 3 conditions:
Be based on the microfacet surface model.
Be energy conserving.
Use a physically based BRDF.



SSAO

light scatters in all kinds of directions with varying intensities so the indirectly lit parts of a scene should also have varying intensities, instead of a constant ambient component. One type of indirect lighting approximation is called ambient occlusion that tries to approximate indirect lighting by darkening creases, holes and surfaces that are close to each other. These areas are largely occluded by surrounding geometry and thus light rays have less places to escape, hence the areas appear darker.

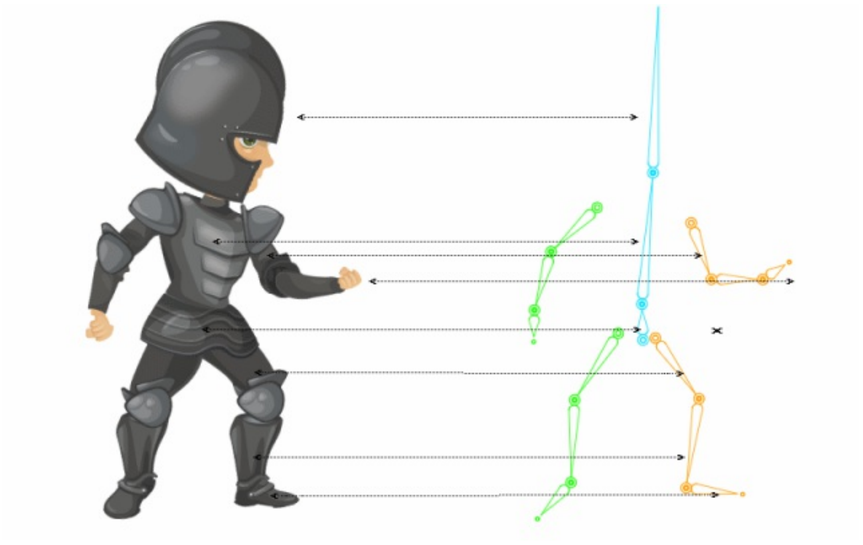
Ambient occlusion techniques are expensive as they have to take surrounding geometry into account. One could shoot a large number of rays for each point in space to determine its amount of occlusion, but that quickly becomes computationally infeasible for real-time solutions. In 2007 Crytek published a technique called screen-space ambient occlusion (SSAO) for use in their title Crysis. The technique uses a scene's depth in screen-space to determine the amount of occlusion instead of real geometrical data. This approach is incredibly fast compared to real ambient occlusion and gives plausible results, making it the de-facto standard for approximating real-time ambient occlusion.



Unit IV - Animation

Skeletal Animation

Skeletal animation or bone-based animation is a technique in computer animation in which the object being animated has two main components: a surface representation used to draw the object (skin, mesh, character) and set of interconnected bones used to animate the surface (skeleton, rig, set of bones).



Skinned Mesh Animation using Matrices

Animating a skinned mesh relies on the principle that scaling, rotating and translating (SRT) a position in space can be represented by a matrix. Further, an ordered sequence of SRTs can be represented by a single matrix resulting from multiplying a sequence of matrices, each representing an individual SRT.

FinalVector = TranslationMat * RotationMat * ScaleMat * vector

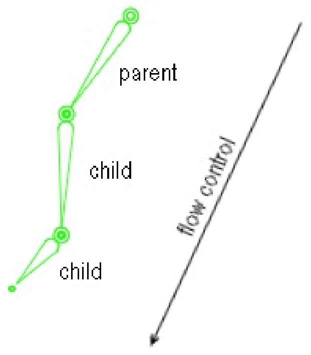
Degree of Freedom

Pitch, yaw and roll are the three dimensions of movement when an object moves through a medium.

Pitch: nose up or tail up.
Yaw: nose moves from side to side
Roll: a circular (clockwise or anticlockwise) movement of the body as it moves forward

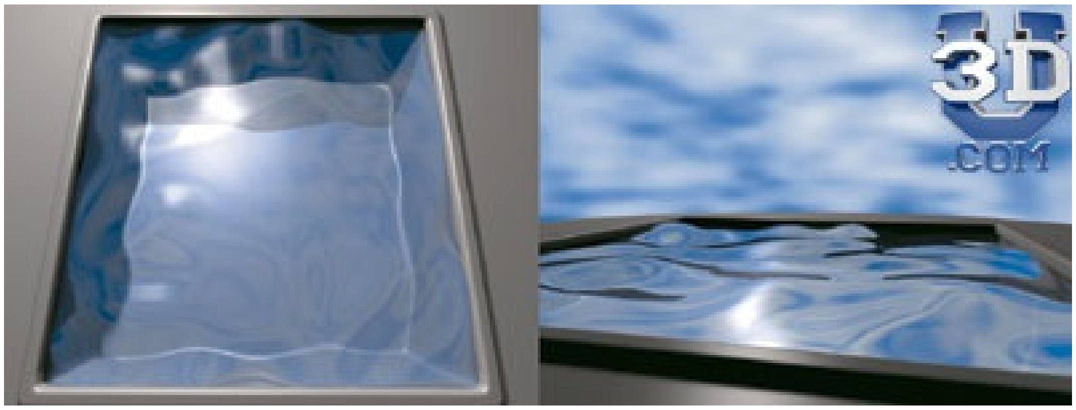
Hierarchy and Bone Weights

A skeleton is a set of bones that can be hierarchical organized. Usually an object has some important joints (vertexes). For instance, for a character the most important joints are the hip, the knee, the neck, etc. A bone connects two already defined joints. Hierarchy means that a a bone belongs to a parent, and each bone can have a child. Affecting a bone also affects all of its children. So, when setting up a skeleton, the first joint is called the root bone. Every subsequent bone will be connected to the root bone either directly, or indirectly through another bone.



Fresnel Effect

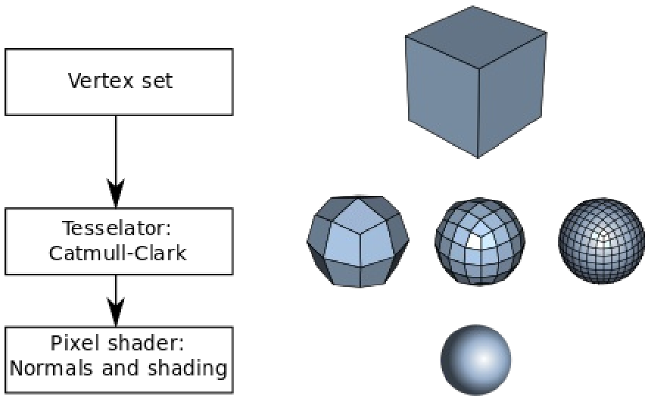
Fresnel Effect is the observation that the amount of reflectance you see on a surface depends on the viewing angle. If you look straight down from above at a pool of water, you will not see very much reflected light on the surface of the pool, and can see down through the surface to the bottom of the pool. At a glancing angle (looking with your eye level with the water, from the edge of the water surface), you will see much more specularity and reflections on the water surface, and might not be able to see what's under the water.



Unit V - Tessellation

Tessellation Stages

Tessellation is used to manage datasets of polygons (sometimes called vertex sets) presenting objects in a scene and divide them into suitable structures for rendering. Especially for real-time rendering, data is tessellated into triangles.



Tessellation Primitive Types

he constructed design is represented by a boundary representation topological model, where analytical 3D surfaces and curves, limited to faces, edges, and vertices, constitute a continuous boundary of a 3D body. Arbitrary 3D bodies are often too complicated to analyze directly. So they are approximated (tessellated) with a mesh of small, easy-to-analyze pieces of 3D volume—usually either irregular tetrahedra, or irregular hexahedra. The mesh is used for finite element analysis.

The mesh of a surface is usually generated per individual faces and edges (approximated to polylines) so that original limit vertices are included into mesh. To ensure that approximation of the original surface suits the needs of further processing, three basic parameters are usually defined for the surface mesh generator:

- The maximum allowed distance between the planar approximation polygon and the surface (known as "sag"). This parameter ensures that mesh is similar enough to the original analytical surface (or the polyline is similar to the original curve).
- The maximum allowed size of the approximation polygon (for triangulations it can be maximum allowed length of triangle sides). This parameter ensures enough detail for further analysis.
- The maximum allowed angle between two adjacent approximation polygons (on the same face). This parameter ensures that even very small humps or hollows that can have significant effect to analysis will not disappear in mesh.

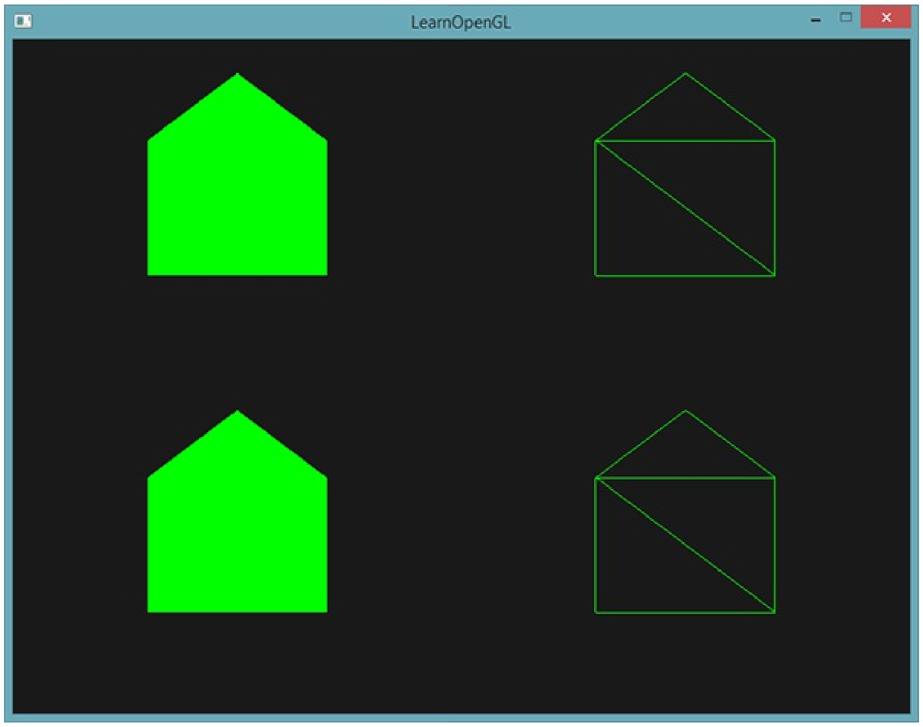
Tessellation Control

As of OpenGL 4.0 and Direct3D 11, a new shader class called a tessellation shader has been added. It adds two new shader stages to the traditional model: tessellation control shaders (also known as hull shaders) and tessellation evaluation shaders (also known as Domain Shaders), which together allow for simpler meshes to be subdivided into finer meshes at run-time according to a mathematical function. The function can be related to a variety of variables, most notably the distance from the viewing camera to allow active level-of-detail scaling. This allows objects close to the camera to have fine detail, while further away ones can have more coarse meshes, yet seem comparable in quality. It also can drastically reduce required mesh bandwidth by allowing meshes to be refined once inside the shader units instead of downsampling very complex ones from memory. Some algorithms can upsample any arbitrary mesh, while others allow for "hinting" in meshes to dictate the most characteristic vertices and edges.

Geometry Shader

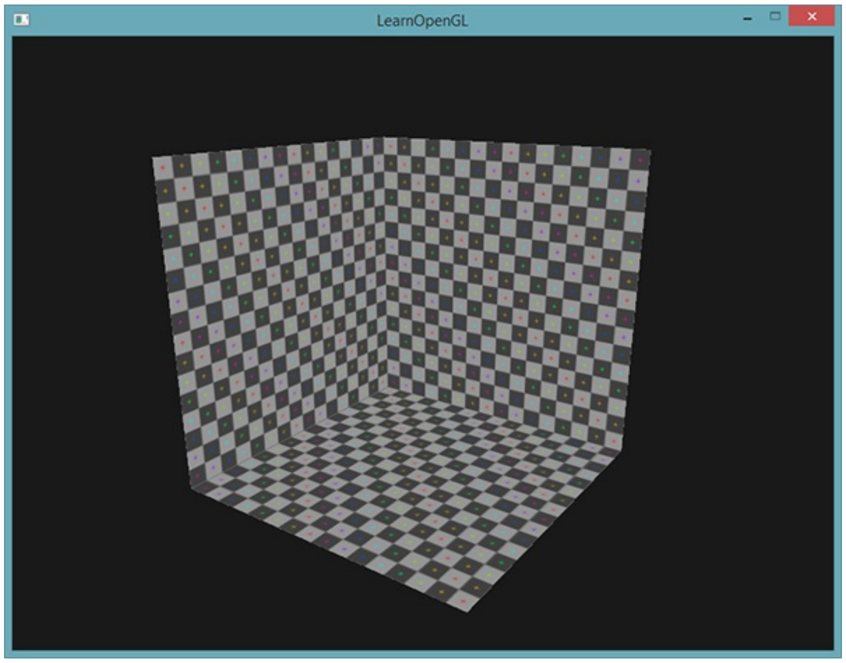
A geometry shader takes as input a set of vertices that form a single primitive e.g. a point or a triangle. The geometry shader can then transform these vertices as it sees fit before sending them to the next shader stage. What makes the geometry shader interesting however is that it is able to transform the (set of) vertices to completely different primitives possibly generating much more vertices than were initially given.

```
#version 330 core
layout (points) in;
layout (line_strip, max_vertices = 2) out;

void main() {
    gl_Position = gl_in[0].gl_Position + vec4(-0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    gl_Position = gl_in[0].gl_Position + vec4( 0.1, 0.0, 0.0, 0.0);
    EmitVertex();

    EndPrimitive();
}
```

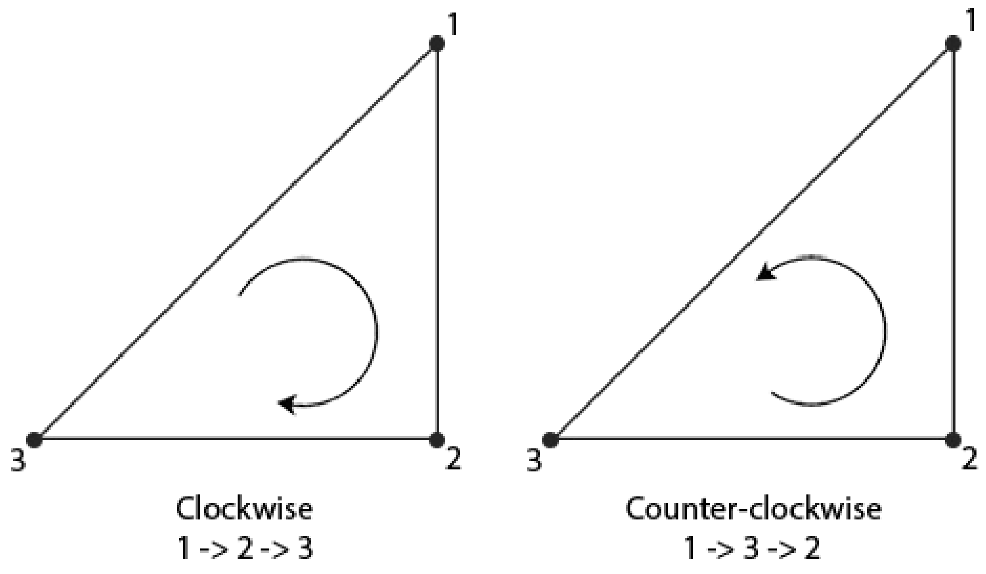


Culling Geometry

You can view a cube from any position and/or direction, but you would never be able to see more than 3 faces. If we could discard those in some way we would save more than 50% of fragment shader runs. OpenGL checks all the faces that are front facing towards the viewer and renders those while discarding all the faces that are back facing saving us a lot of fragment shader calls. We do need to tell OpenGL which of the faces we use are actually the front faces and which faces are the back faces. OpenGL uses a clever trick for this by analyzing the winding order of the vertex data.



When we define a set of triangle vertices we're defining them in a certain winding order that is either clockwise or counter-clockwise. Each triangle consists of 3 vertices and we specify those 3 vertices in a winding order as seen from the center of the triangle.



```
float vertices[] = {
    // clockwise
    vertices[0], // vertex 1
    vertices[1], // vertex 2
    vertices[2], // vertex 3
    // counter-clockwise
    vertices[0], // vertex 1
    vertices[2], // vertex 3
    vertices[1]  // vertex 2
};
```

References

En.wikipedia.org. (2019). Anisotropic filtering. [online] Available at: https://en.wikipedia.org/wiki/Anisotropic_filtering [Accessed 8 Dec. 2019].

En.wikipedia.org. (2019). Per-pixel lighting. [online] Available at: https://en.wikipedia.org/wiki/Per-pixel_lighting [Accessed 8 Dec. 2019].

Zhao, H., Jin, X., Shen, J. and Lu, S. (2009). Fast and Reliable Mouse Picking Using Graphics Hardware. International Journal of Computer Games Technology, 2009, pp.1-7.

Wikipedia. (2019). Pitch, yaw, and roll. [online] Available at: https://simple.wikipedia.org/wiki/Pitch,_yaw,_and_roll [Accessed 8 Dec. 2019].

Solarianprogrammer.com. (2019). OpenGL 101: Matrices - projection, view, model | Solarian Programmer. [online] Available at: https://solarianprogrammer.com/2013/05/22/opengl-101-matrices-projection-view-model/ [Accessed 8 Dec. 2019].

Wiki.polycount.com. (2019). Tiling - polycount. [online] Available at: http://wiki.polycount.com/wiki/Tiling [Accessed 8 Dec. 2019].

En.wikipedia.org. (2019). Alpha mapping. [online] Available at: https://en.wikipedia.org/wiki/Alpha_mapping [Accessed 8 Dec. 2019].

En.wikipedia.org. (2019). Normal mapping. [online] Available at: https://en.wikipedia.org/wiki/Normal_mapping [Accessed 8 Dec. 2019].

En.wikipedia.org. (2019). Projective texture mapping. [online] Available at: https://en.wikipedia.org/wiki/Projective_texture_mapping [Accessed 8 Dec. 2019].

Opengl-tutorial.org. (2019). Tutorial 14 : Render To Texture. [online] Available at: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/ [Accessed 8 Dec. 2019].

En.wikipedia.org. (2019). Multisample anti-aliasing. [online] Available at: https://en.wikipedia.org/wiki/Multisample_anti-aliasing [Accessed 8 Dec. 2019].

En.wikipedia.org. (2019). Heightmap. [online] Available at: https://en.wikipedia.org/wiki/Heightmap [Accessed 9 Dec. 2019].

Technologies, U. (2019). Unity - Manual: Global Illumination. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/Manual/GIIntro.html [Accessed 9 Dec. 2019].

En.wikipedia.org. (2019). Shadow mapping. [online] Available at: https://en.wikipedia.org/wiki/Shadow_mapping [Accessed 9 Dec. 2019].

Gsp.humboldt.edu. (2019). Atmospheric Scattering. [online] Available at: http://gsp.humboldt.edu/OLM/Courses/GSP_216_Online/lesson2-1/scatter.html [Accessed 9 Dec. 2019].

En.wikipedia.org. (2019). Lens flare. [online] Available at: https://en.wikipedia.org/wiki/Lens_flare [Accessed 9 Dec. 2019].

Marionette Studio. (2019). Skeletal Animation - Animate objects using hierarchical bones. [online] Available at: https://marionettestudio.com/skeletal-animation/ [Accessed 9 Dec. 2019].

GameDev.net. (2019). Skinned Mesh Animation Using Matrices. [online] Available at: https://www.gamedev.net/articles/programming/graphics/skinned-mesh-animation-using-matrices-r3577/ [Accessed 9 Dec. 2019].

3drender.com. (2019). Fresnel Effect. [online] Available at: http://www.3drender.com/glossary/fresneleffect.htm [Accessed 9 Dec. 2019].

En.wikipedia.org. (2019). Shader. [online] Available at: https://en.wikipedia.org/wiki/Shader#Tessellation_shaders [Accessed 9 Dec. 2019].

En.wikipedia.org. (2019). Tessellation (computer graphics). [online] Available at: https://en.wikipedia.org/wiki/Tessellation_(computer_graphics) [Accessed 9 Dec. 2019].