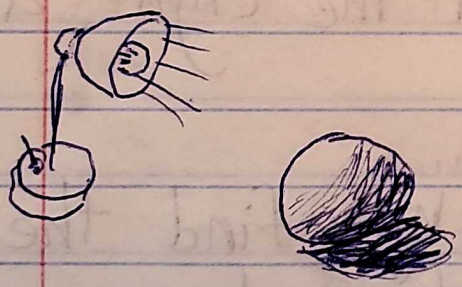
- We need a way to read from a .obj file.
- The OBJLoader class solves this problem.

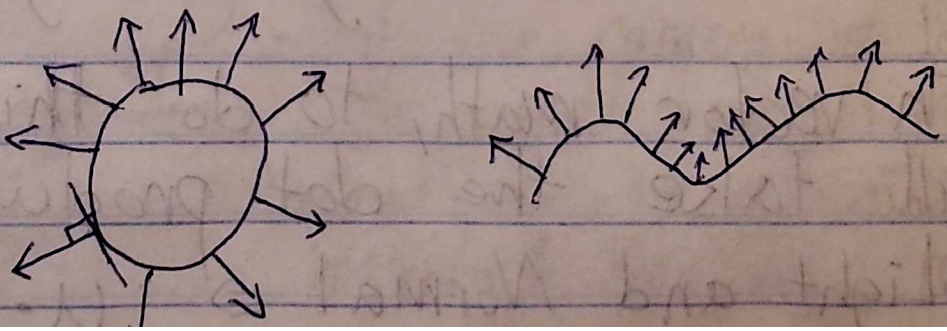
Light:

- I don't need to explain how light works as that is well known, however I will explain that the side of an object that faces the light is more lit up.



- I'm not a good artist but you can see that principle with this drawing.

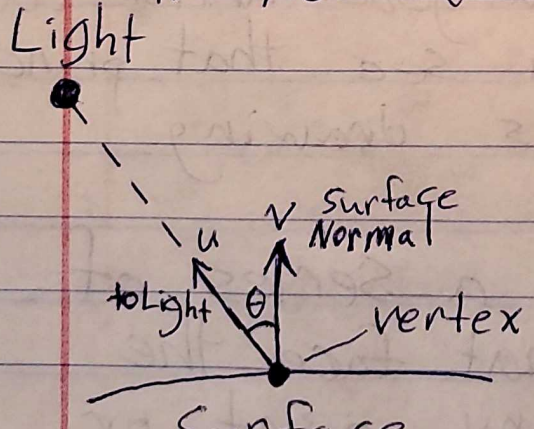
- Each surface has a series of normal vectors that face the same direction as any point, or they are orthogonal to a point on the surface.



- Luckily, the .Obj files have all our model's normal vectors and we just have to load these into our VAO.

VAO Attribute Lists (VBOs)
0 - positions
1 - texture coords
2 - Normals

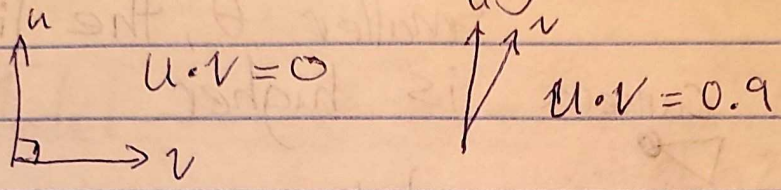
Light Math: - Now we have Normals we can calculate light based on the angle between the light and the normal vector



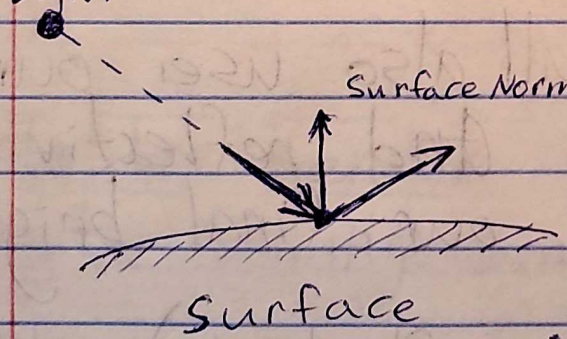
- First we find the "toLight" vector.
- Second we find the angle between the two vectors because this will tell us how strong the light needs to be.

- In vector math, to do this we will take the dot product of toLight and Normal ⇒ $u \cdot v$

- Instead of using degrees or radians of an angle, we will simply use the resultant dot product value. the closer this value is to 1, the brighter the light. value of 0 is almost no light.



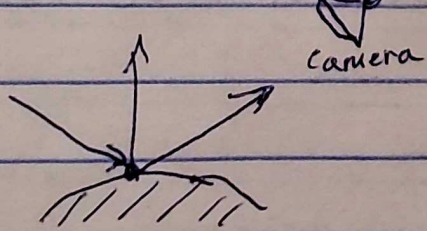
Specular Lighting: Gives objects a shiny/reflective look. It is added onto our previous lighting. (diffused)



- The light is reflected based on the surface's reflectivity

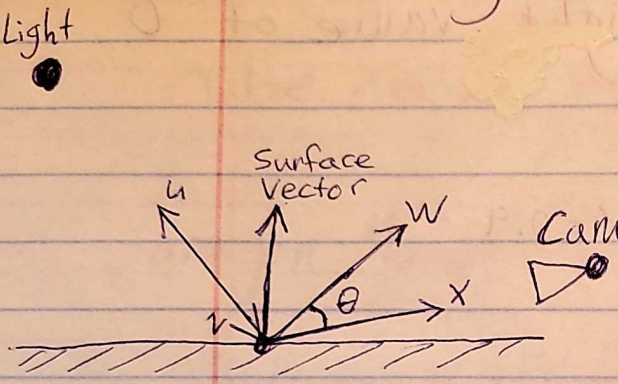
high reflectivity = Lots of reflected light

- Specular lighting also depends on the angle of the camera



- If the camera is looking at the reflected light then it would see it, otherwise it wouldn't see any reflected light.

Shine damping is a variable for how close the camera needs to be to the reflected light to see any change in brightness.



u = to Light vector
v = from Light vector
w = reflection vector
X = to camera vector

- when w and X have a smaller θ, the light reflected is higher.
- well take another dot product of w • X so as to calculate reflected light (brightness).
- we will also use our damper variable and reflectivity variable to alter our final brightness.

$((brightness)^{damper} \times reflectivity) + diffused\ Light$

- GLSL has a reflect function. it takes in a light direction vector (from Light vector) and a surface vector.

- HashMap in MasterRenderer:

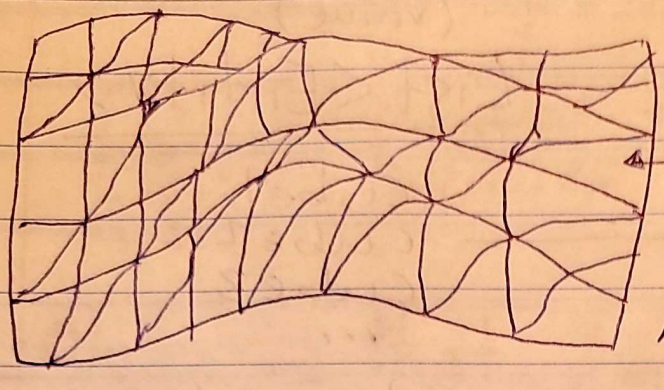
(key) TexturedModel	(value) List<Entity>
cubeModel	cube1, cube2, cube3, ...
dragonModel	dragon1, dragon2
stallModel	stall1

- The TexturedModels are mapped to a list of entities. (Entity objects)

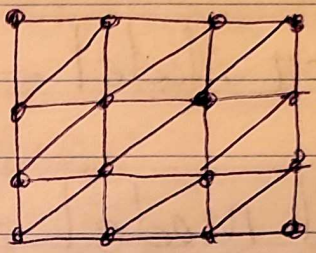
- MasterRender optimizes our code but running similar code less.

Terrain:

-terrain mesh of texture triangles

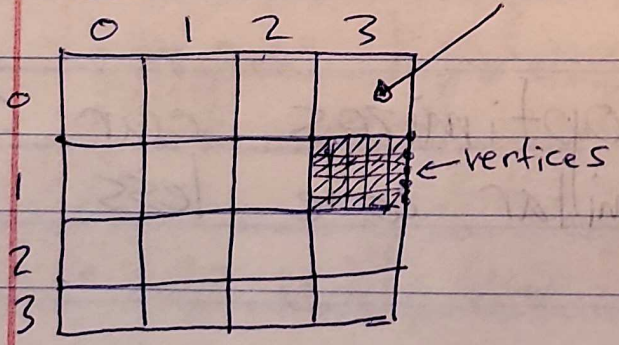


each triangle has a texture.



- terrains are equally spaced vertices (grid)
- Each vertex has its own height, creating the terrain.

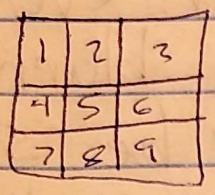
world:



- The world will be organized into a grid of terrain tiles, each with a set amount of vertexes.
- Each tile will be the same size.

- Terrains can become very advanced so we need to separate entity rendering from terrain rendering. (This means whole new shaders)

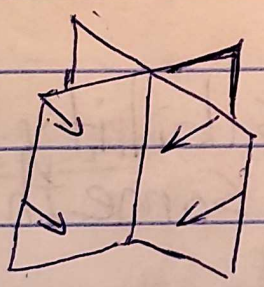
Tiles: splitting up the terrain to hold multiple textures.



- repeated texture over all the squares.

- Objects that have transparency or two conflicting facing faces have messed up light-values.

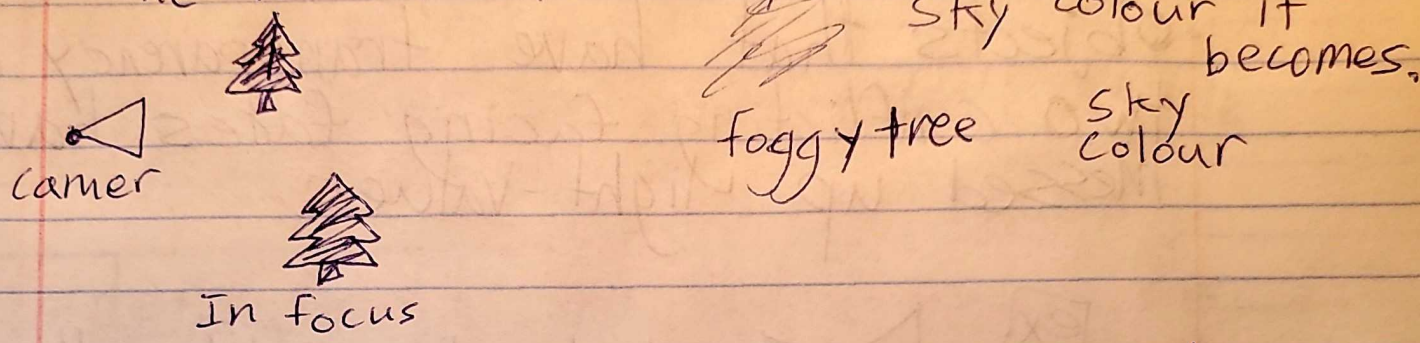
ex



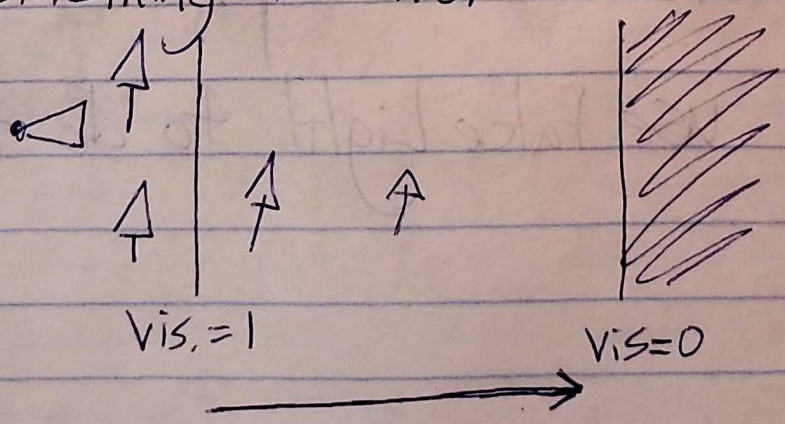
- A texture like this has conflicting vectors.
- In order to get proper lighting, we will make all the vectors face up.

- we will use FakeLight to do this.

- Fog: we want to implement "fog" so that the scene Camera can't see clearly forever. eventually the distance becomes blurry (foggy) and can no longer be seen.
  - The farther it is away, the more like the sky colour it becomes.

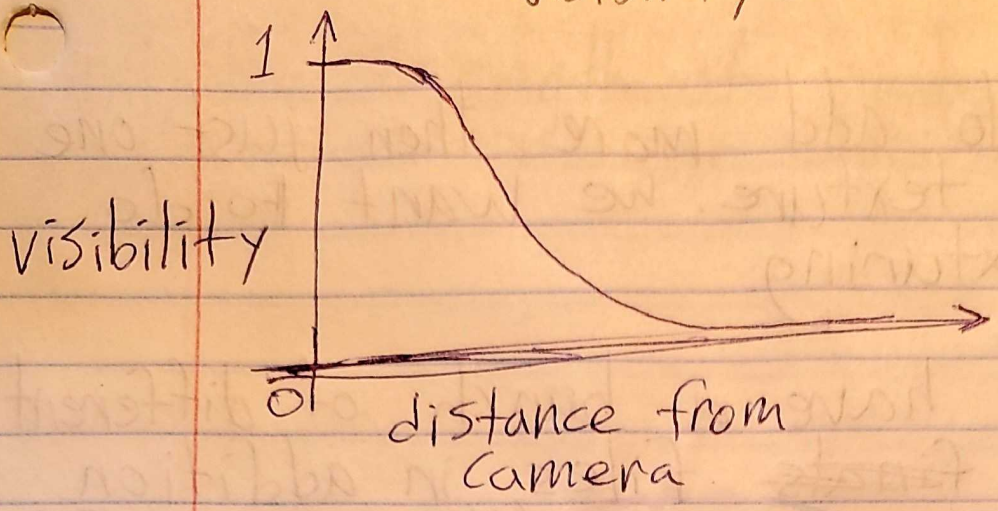


- We will use a visibility variable to calculate if something is visible.
- Visibility is determined by the distance something is from the camera



- we will use an exponential function to ~~much~~ move from the visibility = 1 line to the Vis. = 0 line, to calculate all the values in between.

$$visibility = e^{-(distance \cdot density)^{gradient}}$$

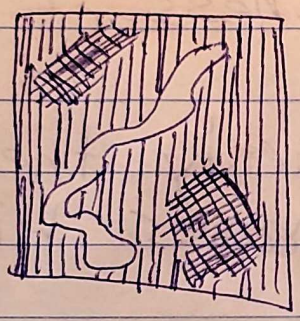


- Visibility is the value of how visible an object is.
- density is the thickness of the fog.
- gradient is how quickly the visibility decreases with distance

- Code for this starts in the vertex shader.

- we want to add more then just one ground texture. we want to do multi-texturing.

-we will have a bunch of different texture ~~finals~~ files, in addition to having a single "blend map" file.
-This tells the code where to render our ground textures.



O = path textures
= dirt textures
|||| = grass textures
etc.

- will implement this in the fragment shader because that is where each pixel's color is determined.

-usually we map a single pixel's color to a single texture's colour.

-Now we will map that pixel to a pixel on each of our textures.
-we will blend these pixels to create our final colour.