OBJ Mapping in DarkPlaces engine

Hello fellow developers. If you are developing a standalone mod or a game using modern quake source port or just want to move away from quake engines this information will be useful.
So, I say, you should not use BSP for your levels, instead use OBJ. The first question you may ask is what is wrong with the BSP format. Lets look:

	Q1 BSP	OBJ
Limits	Tiny bounds, low polygon limit, no complex geometry is allowed.	The limit is your hardware
Textures	Build-in. 254 color palette, only divisible by 8px. Can be replaced with external textures. Requires a WAD.	External textures
Occlusion culling	VIS Process. Hard and time consuming. Map should be completely sealed (leak check). Requires placing special entities. Takes some time.	Manual culling or distance based culling done in QC
Compiling time	Takes some time.	Instant (~0.5 sec)
Lighting	Nice build in lightmaps that affect dynamic entities and have cool bounce and ambient lights. Baking takes a short time.	Dynamic lights or lightmaps baked directly to textures. Lightmaps do not affect dynamic entities. Blender allows you to edit lights in real time without baking and has more light features.
Tools	Various tools: map compiler, wally, compiler GUI, etc.	Any modern 3D software. (just Blender). Various AI tools and generators.
Engines	Supported by all engines.	Only a few.
Level editor	Trenchbroom/J.A.C.K	Any modern 3D software. (Blender)
Polycount	Generates thousands of unwanted faces.	At least 3 times less vertices compared to any BSPs
Brain damage	Results in permanent brain damage and contamination.	Results in learning modern software, developing skills.

But it is not all about BSP problems, it is about what modern modeling techniques can achieve. Modern software has tons of software like procedural materials that do not repeat themself, linked meshes, prefabs, randomization, modifiers, visual light editor, various generators, AI tools and so much more!

My first OBJ map. Made in Blender. DarkPlaces engine.

It is true that OBJ is not perfect. It has very serious issues. It will not fit for any game, it depends on game style and other stuff. And we will discuss that later, but generally you will be able to achieve much better results in much shorter time. Because you don't need to waste time doing stupid shit like packing your textures inside WAD, converting geometry into brush entities, working with outdated software and so on. If everyone will start using OBJ all these issues will be fixed in a matter of weeks.

If you know how to use BSP, if it suits your needs, if you want only to make PS1 styled maps it is fine, but think about that like about a car. You have an old car you are very comfortable with. It gets the job done, you like it, but newer cars are just faster. They are beautiful. They have other features. And it doesn't matter how much effort you put inside updating your old car it will never reach modern cars. So you sit in your community discussing old cars with your friend, but life goes on. And the modern generation knows nothing about old cars. They think about you as a fool who wastes time on rusty metal.

Understanding

Let's start with beginning and understanding how .obj maps work. Download DarkPlaces. You may be aware of commands sv_saveentfile and modeldecompile.
sv_saveentfile - saves all entities on the current map to .ent file inside your game folder. Why do you need it? If this file exists while loading map it will override map's entities so developer can edit entities without compiling map. This is useful for patching and fixing maps after it was released or adding entities to make the map work for you.
modeldecompile - converts and exports selected model in .obj format. What modeldecompiling has to do with maps?
Let's start darkplaces and load any map. For example I will use E1M1 from Quake. After you load on map type sv_saveentfile in the console. You should see the .ent file with map name appear inside the "maps" folder (id/maps/e1m1.ent). Let's view it:

```
{
"worldtype" "2"
"sounds" "6"
"classname" "worldspawn"
"wad" "gfx/base.wad"
"message" "the Slipgate Complex"
}
{
"classname" "info_player_start"
"origin" "480 -352 88"
"angle" "90"
}
```

The first block spawns the world. Second defines player spawn point. You can edit or add properties just like in the .map file.

Now let's try another command. Open the console and type *modeldecompile maps/e1m1.bsp* Check the map folder again. 2 files (e1m1_decompiled.obj and e1m1_decompiled.mtl) should appear. Now lets rename our ent file (e1m1.ent) to match the model name (e1m1_decompiled.ent).
Now execute *map e1m1_decompiled.obj* in the console to load the map. You should now be able to play the map except it has no textures (if the game is too dark, enable dynamic lights). Normally quake stores textures inside a BSP file (which is stupid idea btw). To use external textures you need to put .tga (or png/jpg if you have needed a dll library) inside the game folder or textures folder (id1/textures/).
To get quake textures easy you can download
https://www.moddb.com/games/quake/addons/quake-mini-hd-build

Depending on your build you may be required to specify a full path for the texture.
To fix textures you need to edit the .obj file (mtl is not rly needed, you can remove it). Find usemtl line (smth like usemtl tech08_1). Second part is the name of our texture file relative to the game folder. Replace "usemtl " with "usemtl textures/". This should work for any engine.

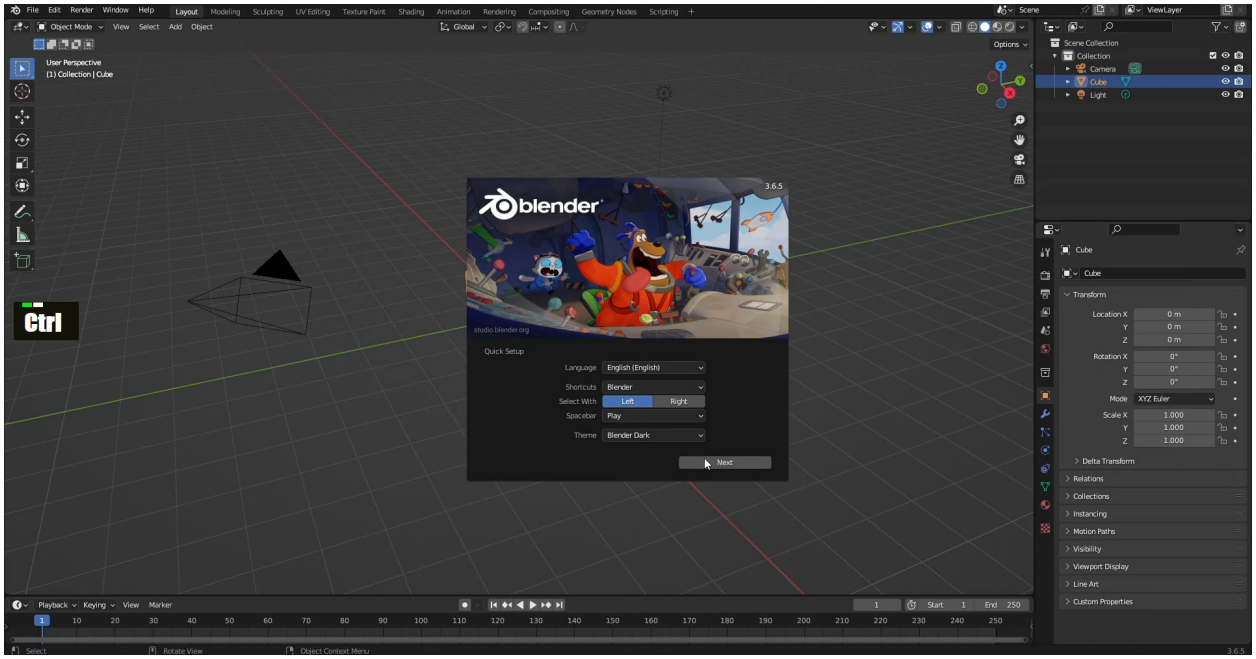
Getting started with OBJ mapping

There are many model editors that support .OBJ editing. The most popular 3d model editor is Blender. It is free, user friendly, easy to learn and professional enough, has tons of addons and easy scripting syntax. Learn blender and don't even think about Houdini or 3ds Max or some other fancy software. Believe me, it is not worth it. Especially for quake modding.

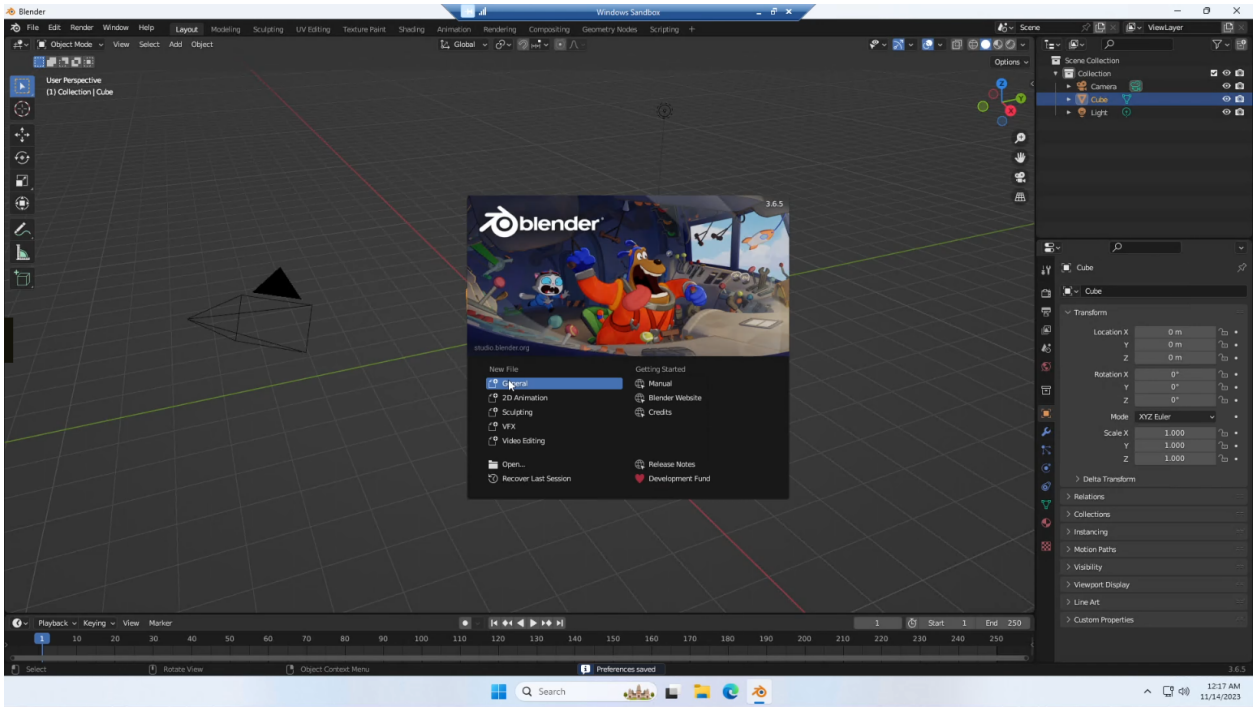
I am not going to post the entire blender tutorial because it will take a few pages. You can find anything online. Here is very basic quickstart guide:

Blender Quickstart guide

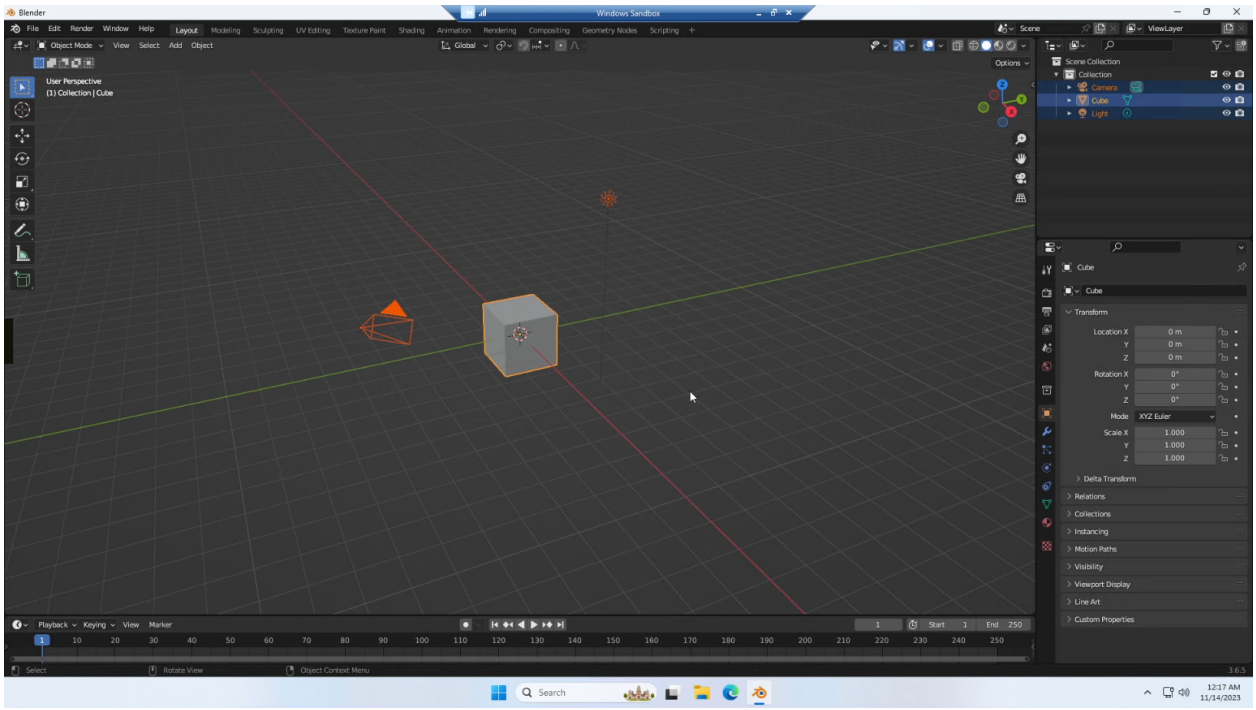
Start by downloading and installing Blender.
Next after you launch it you will see an initial setup window. You can just skip it.



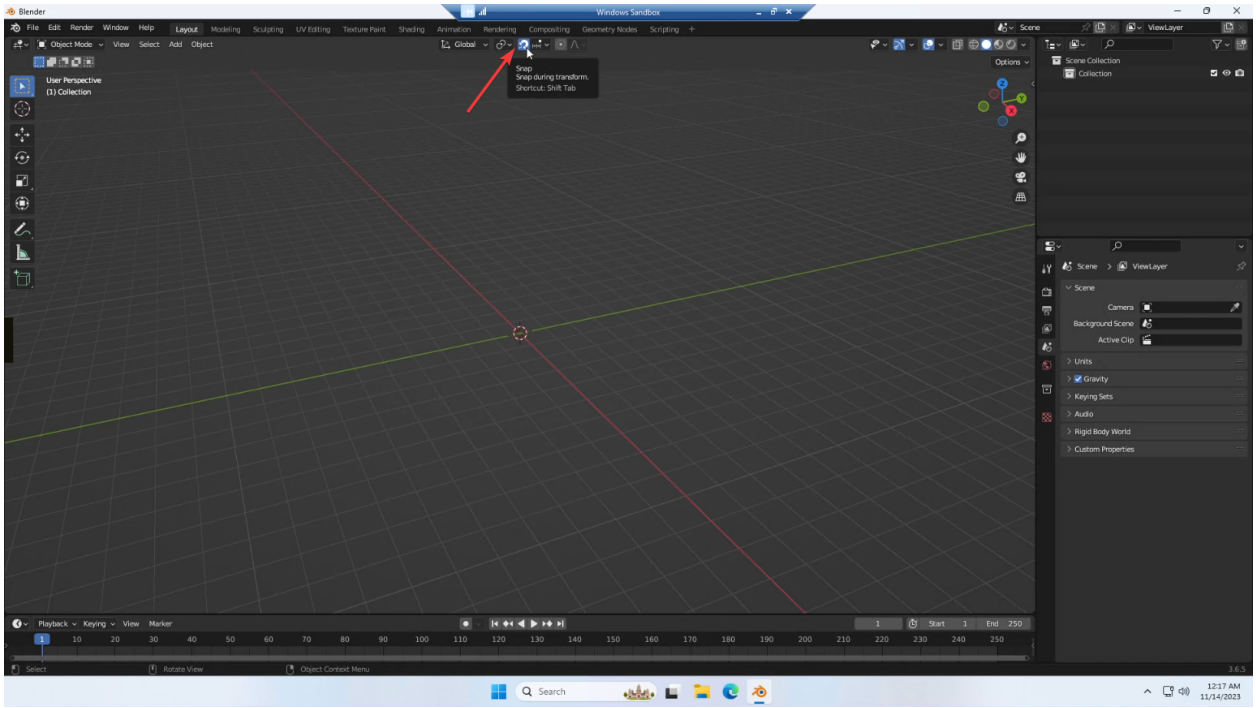
After that click on New file > General



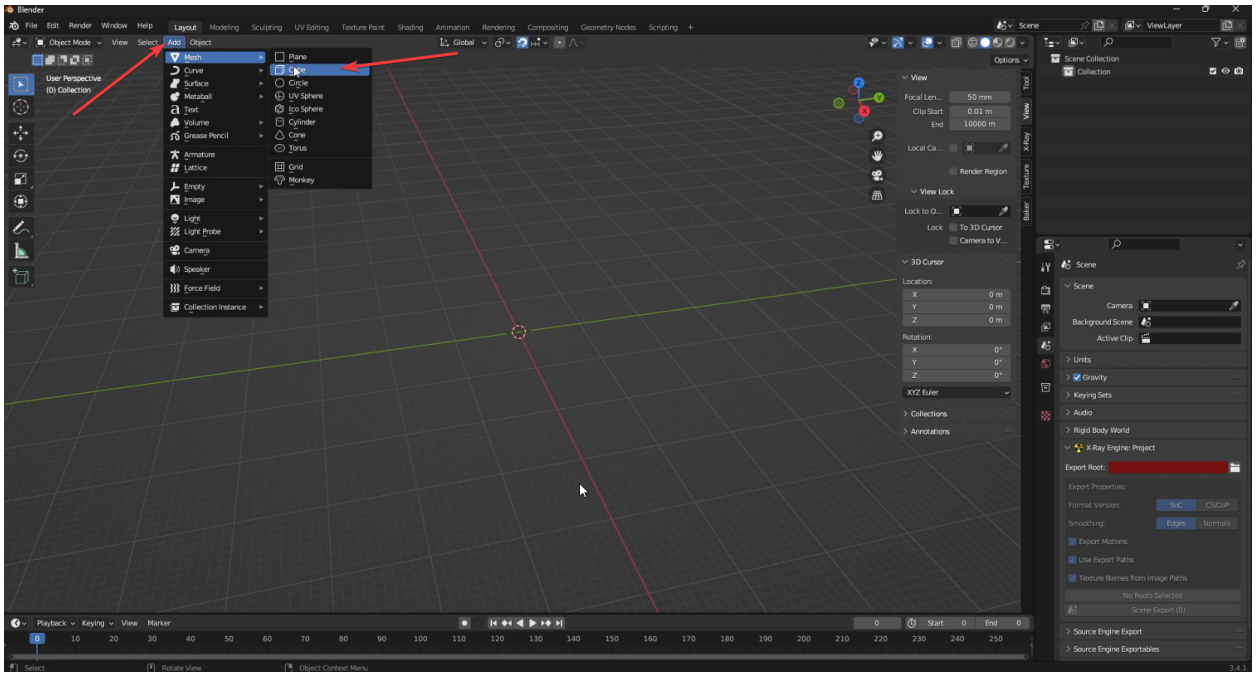
It will create a default scene with a cube, camera and a light. We don't need all that. You can select it and delete it.



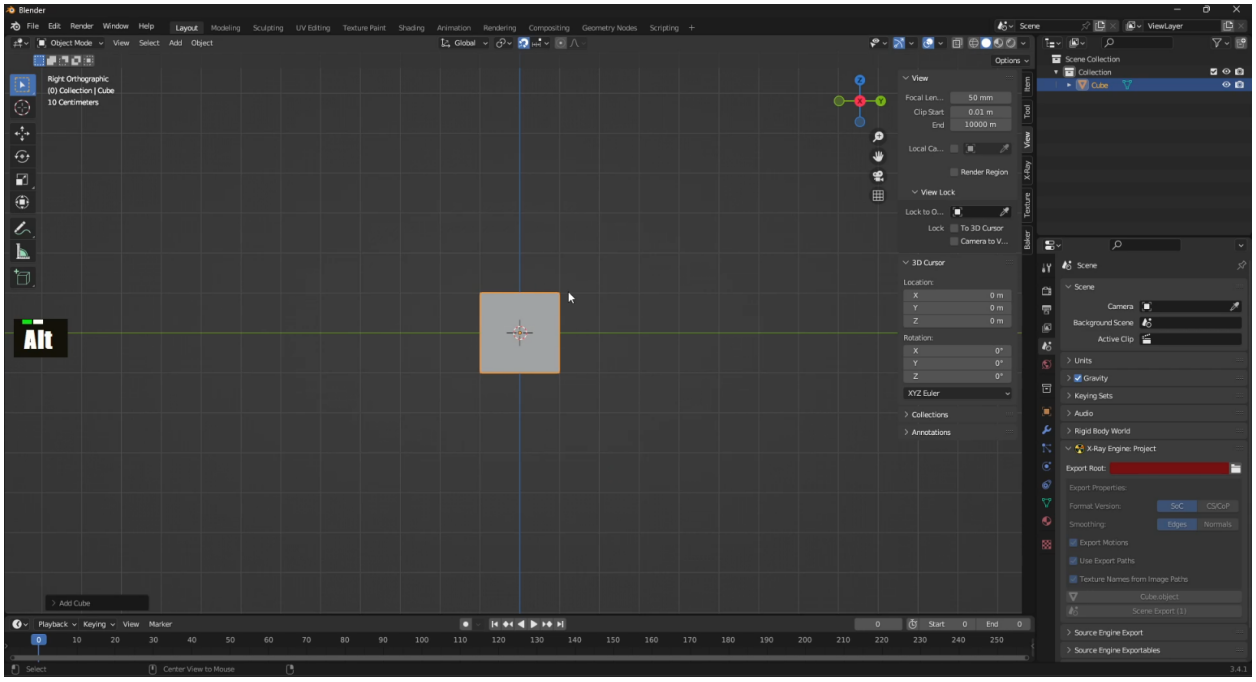
I recommend you enable SNAP mode (magnet icon on top center). This will make all vertices snap to grid if you edit them. Each base cell is 1 meter cube.You can hold CTRL to temporarily unsnap. Zoom in to increase the grid.



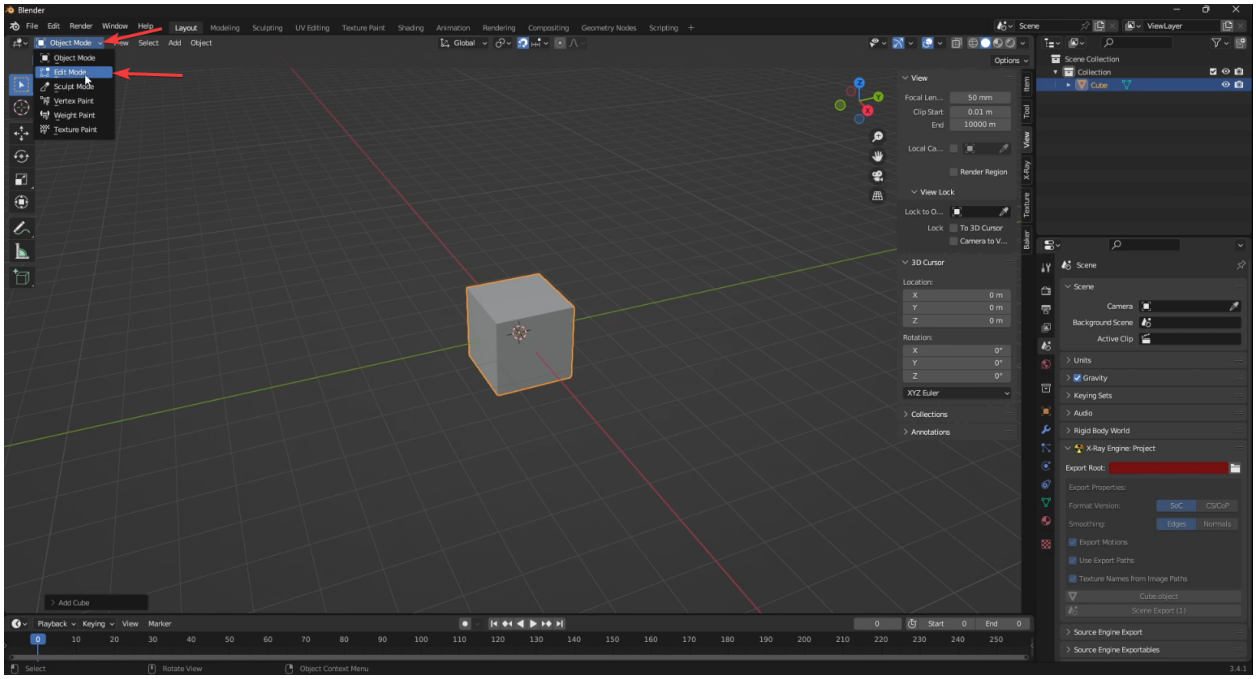
Now let's add our first object. Let’s add the cube since it is very similar to a brush.



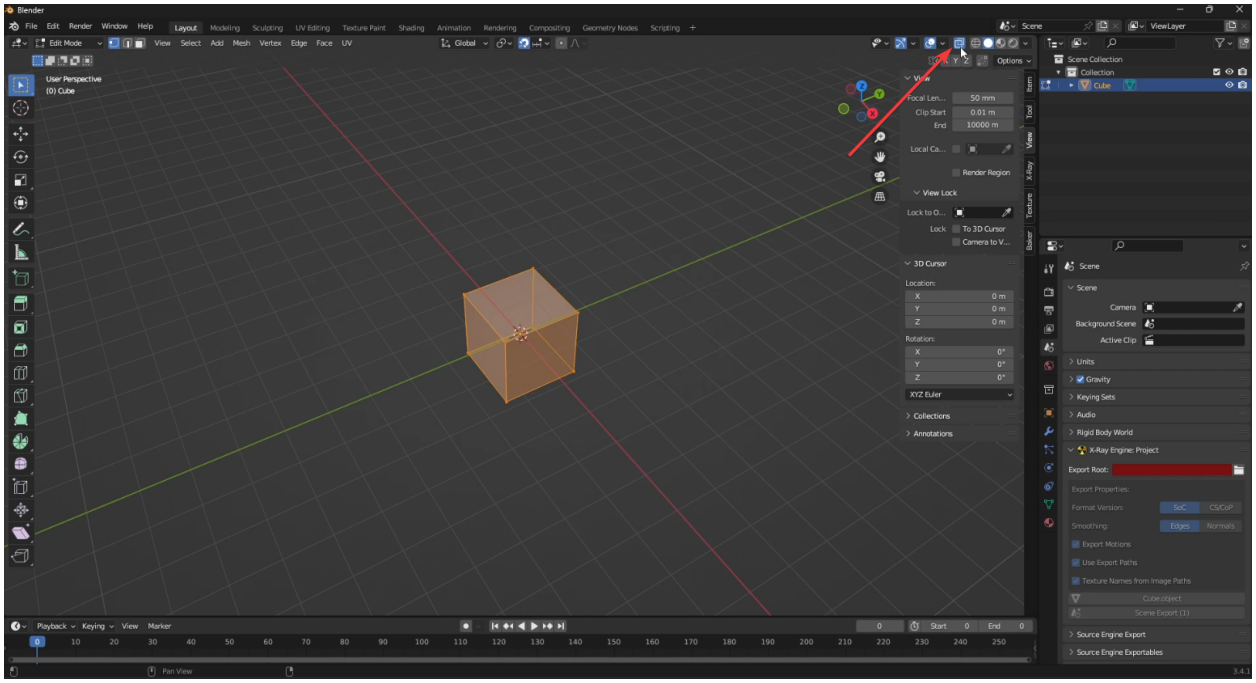
If you hold the Mouse Wheel button you can rotate your view. If you hold both ALT key and MWHEEL you can assign your view to axis.



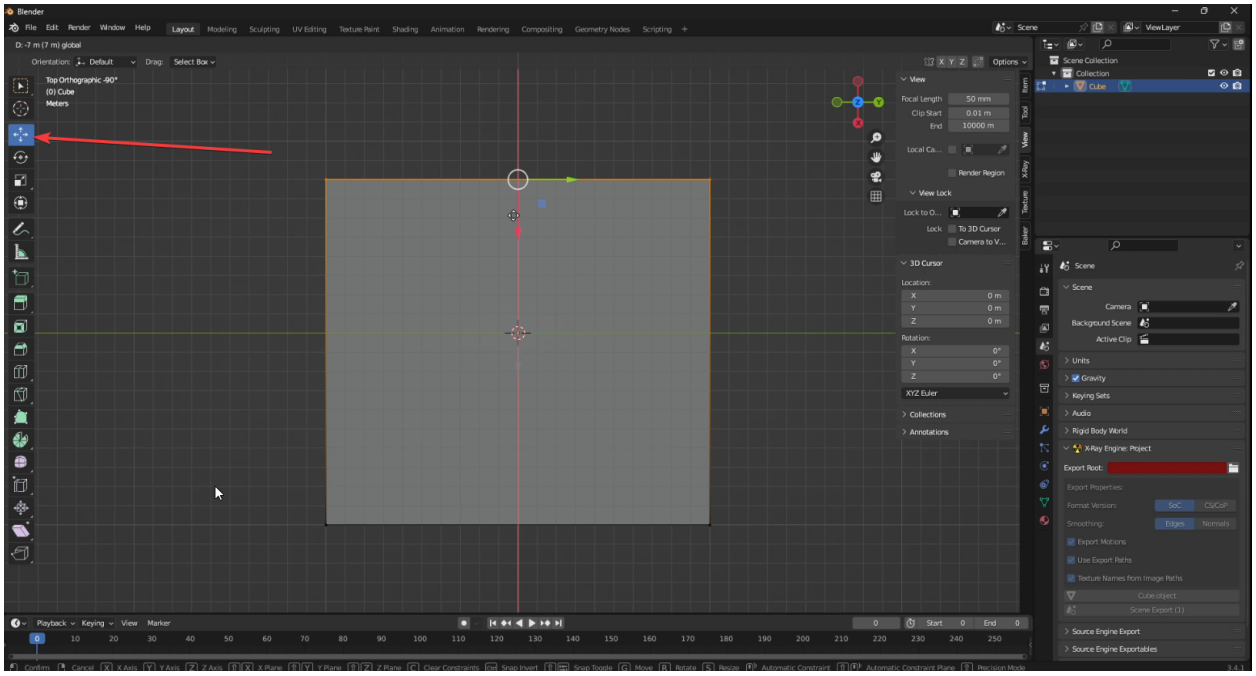
Now let's edit our cube. To do this click on mode combobox and select edit mode or simply select an object and press TAB key.



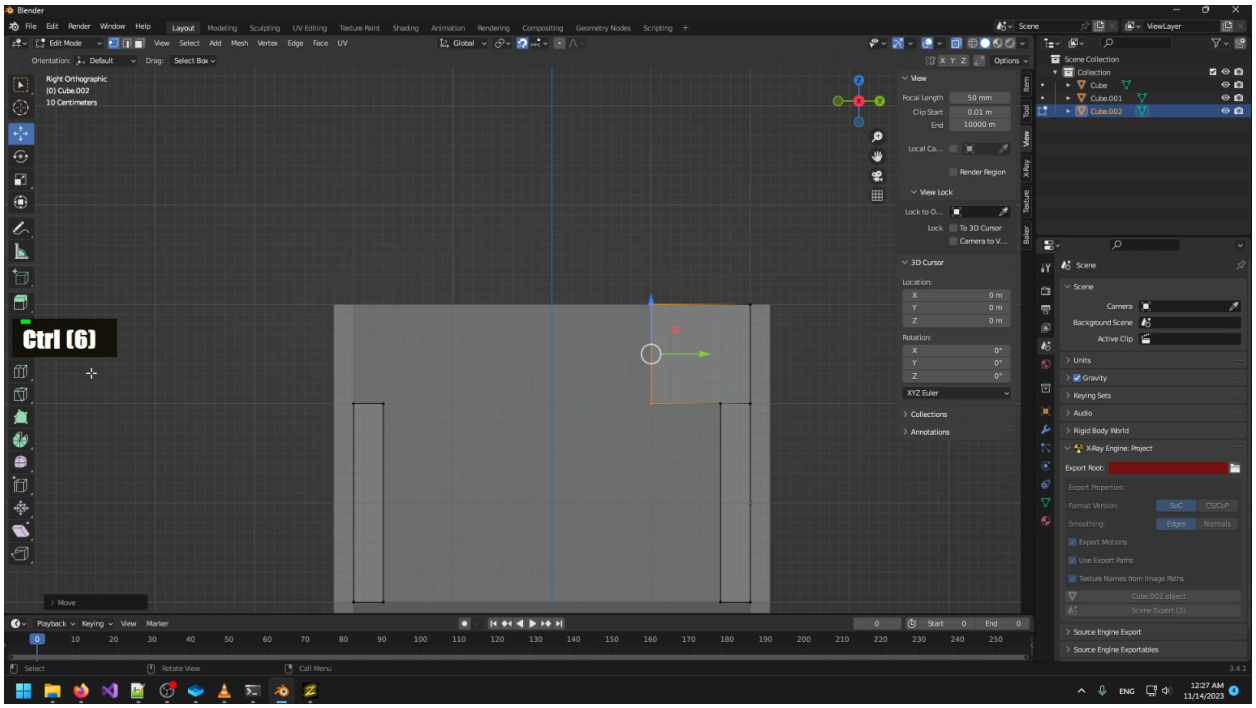
Now we can see, select and edit our vertices. For comfort let’s enable X-Ray vision by clicking the X-Ray icon near viewport shading options.



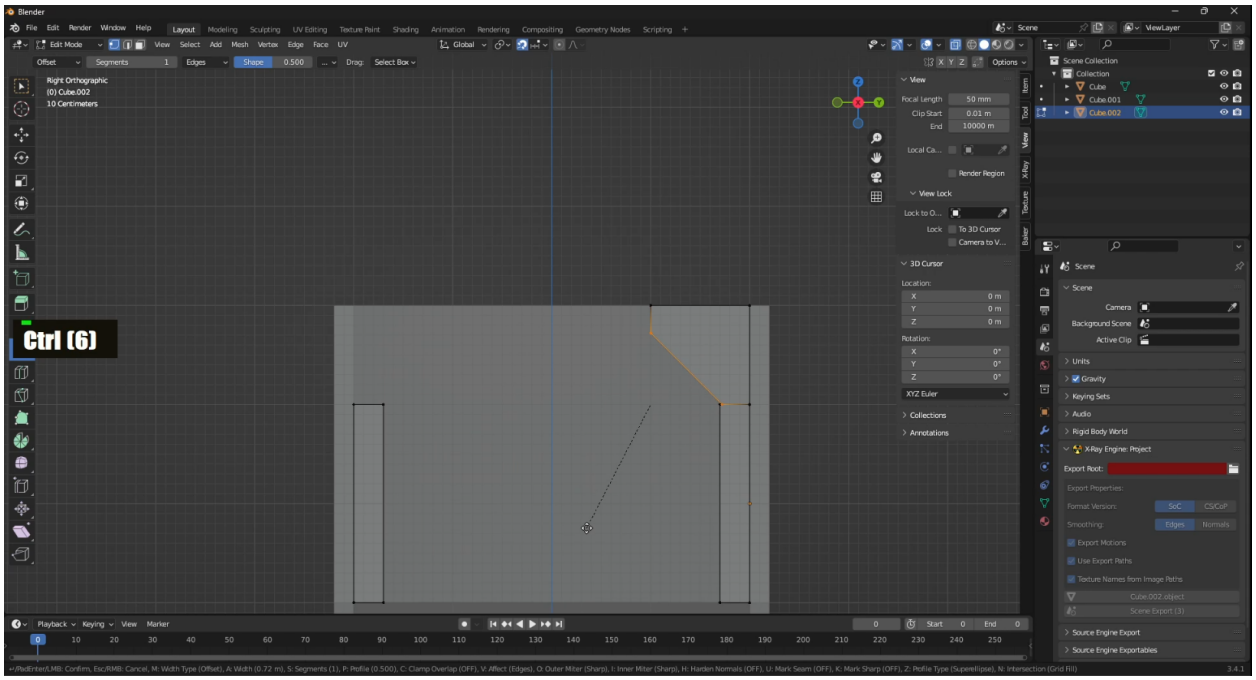
Let's form a base plate. Select vertices that you want to move. Hold CTRL to deselect. If you want to switch the selection tool press W key. Click on the move tool to move verts. Do the same for walls.



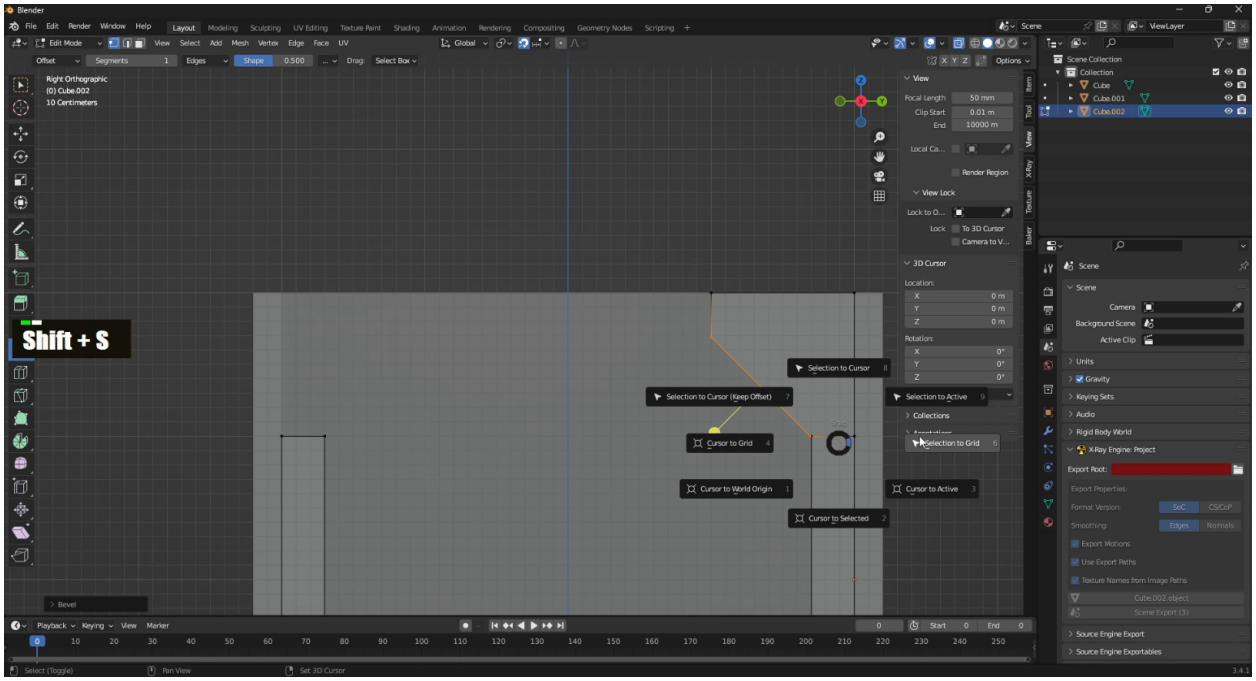
Now let's create something more complex - a pillar. Create a parallelepiped. To Quickly add another parallelepiped you can duplicate it. Select one vertex and press CTRL + L . This will select all verts that are linked (connected) to selected. Now lets press SHIFT + D to duplicate the parallelepiped and place it on top of the original one.



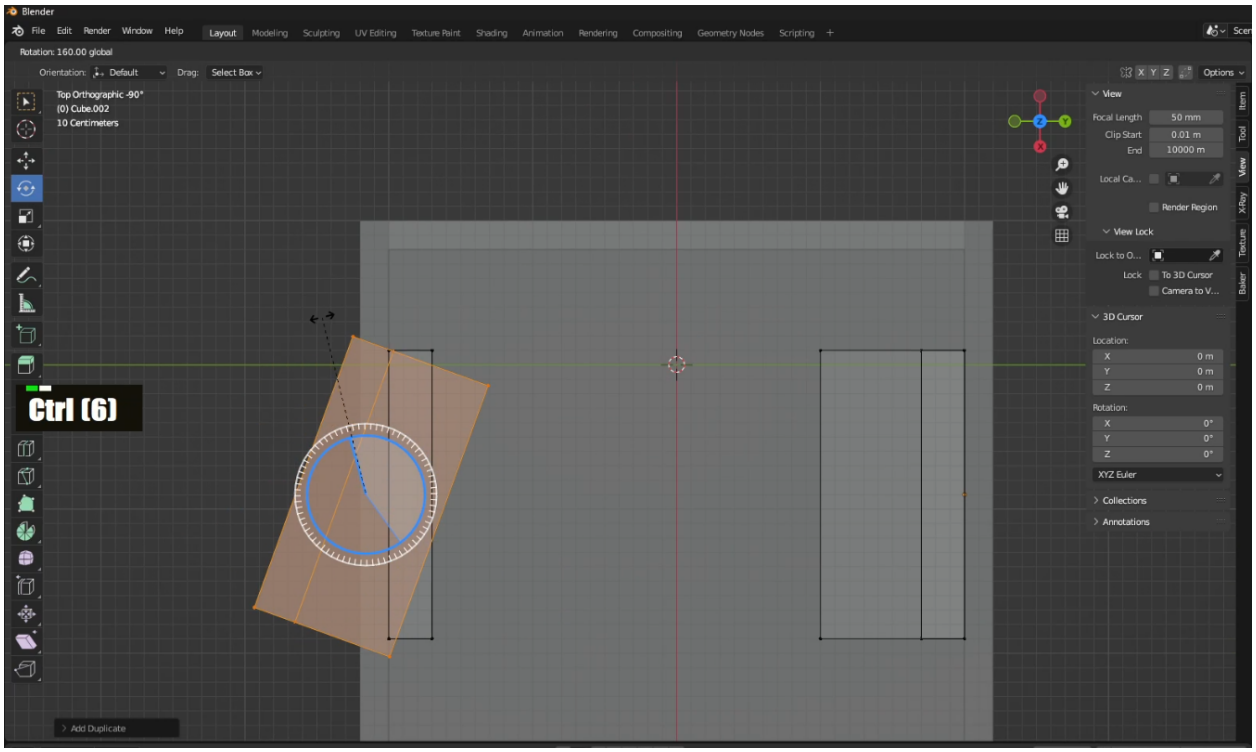
Now I want to show you a very cool Blender tool. It is called Bevel. Select 2 edge verts (or more) and click on the bevel icon (box with sliced corners). Now drag appeared to form something like on the screenshot. You can scroll the mouse wheel to adjust the amount of cegments to form a perfect arch.



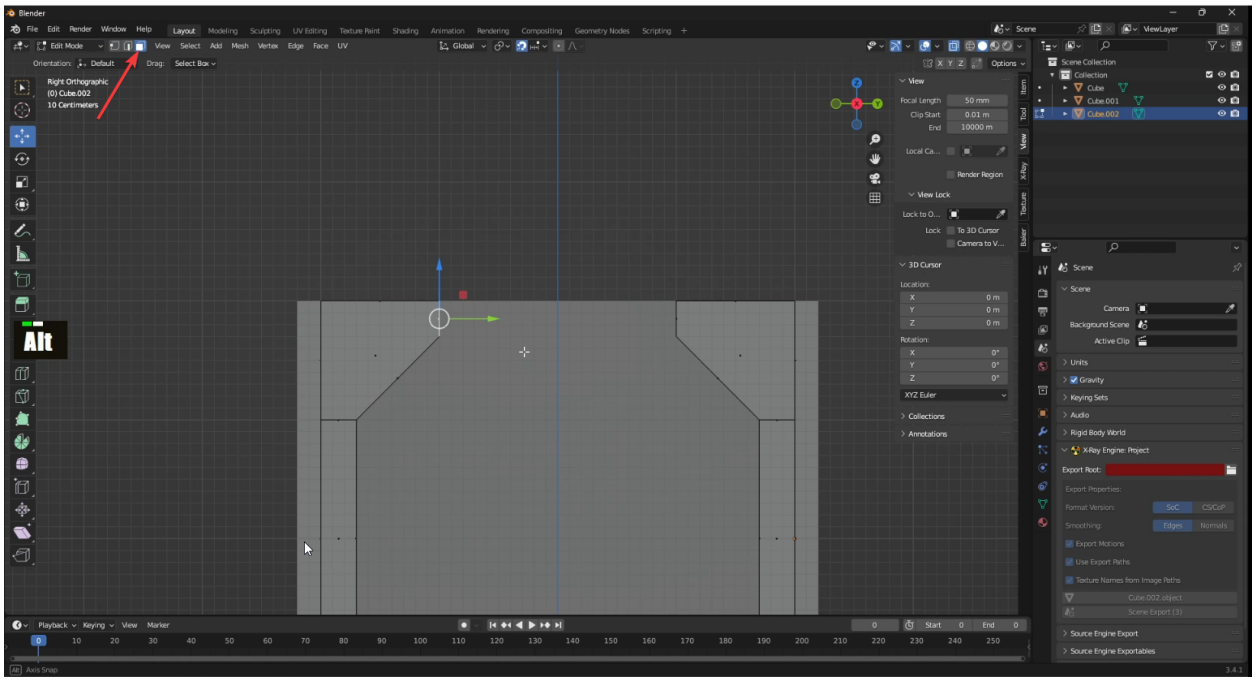
After you are done you may see that vertices are not on the grid. To fix that select them and press SHIFT +S and then select Snap to grid.



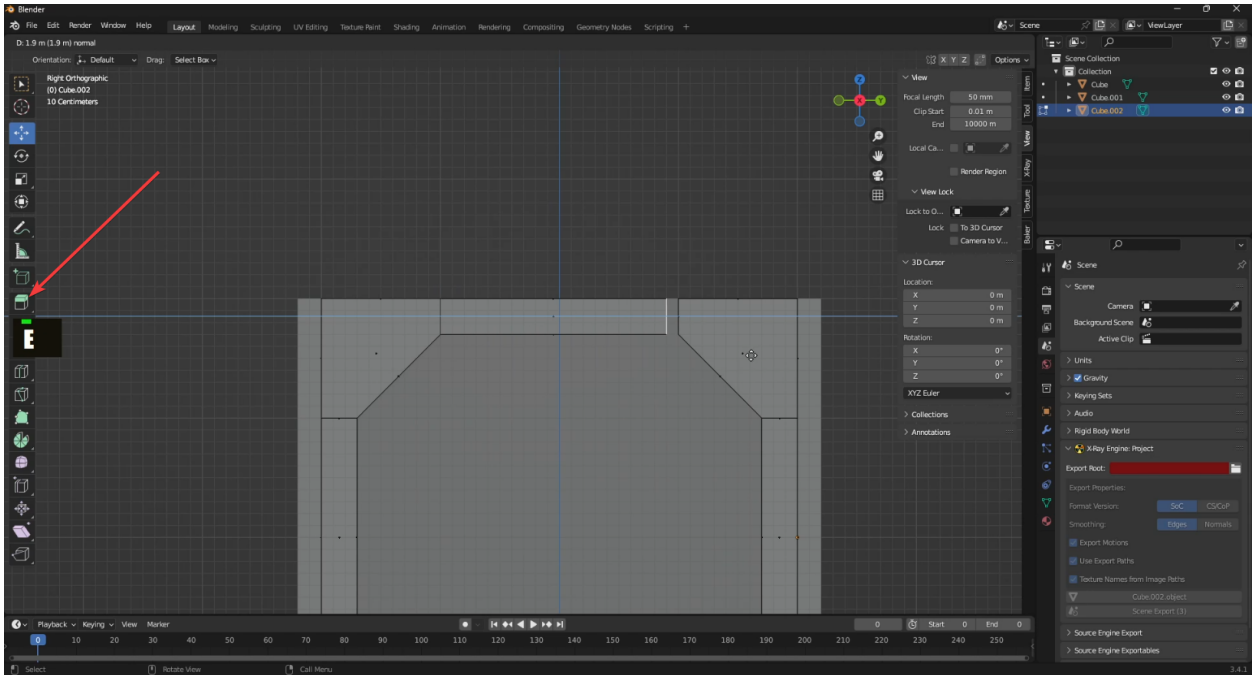
Now let's copy this pillar. Select all verts and press CTRL + L (select all linked verts in case we miss something). Press Shift+D to duplicate. Now let's rotate it using the rotate tool. Hold CTRL while rotating. Don't forget to snap to the grid after rotating.



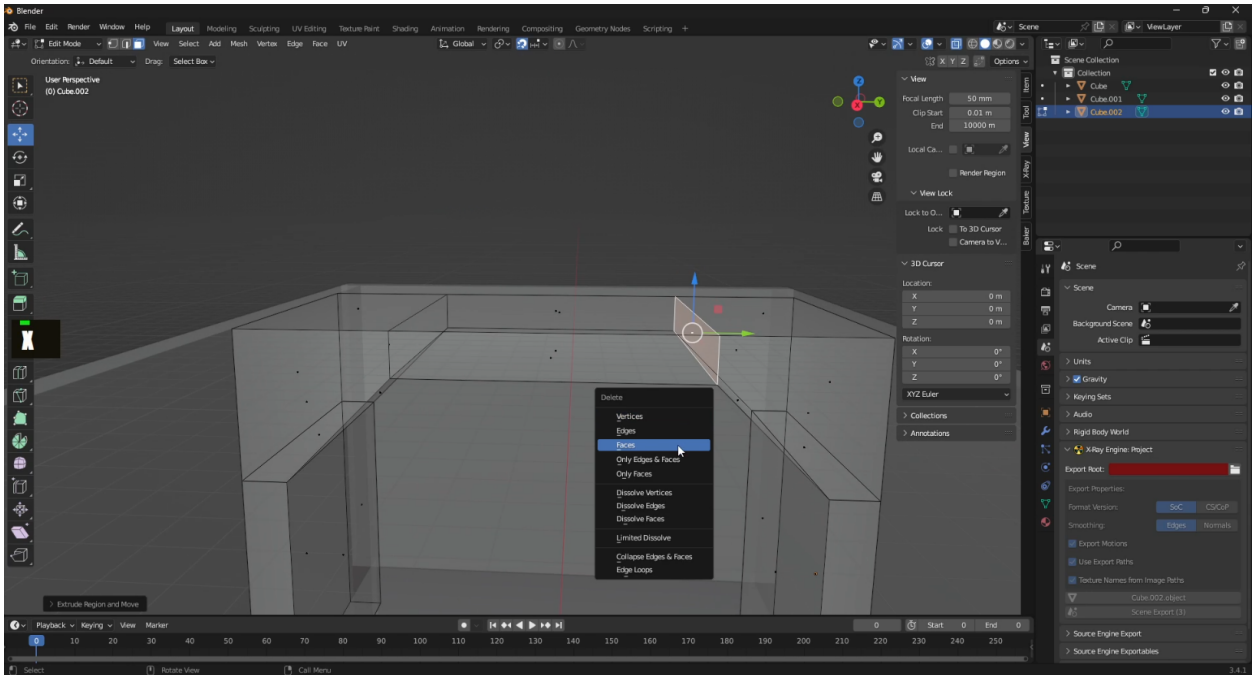
Now I will show you another cool tool. It is called Extrude. First let's switch to face mode. To do this press 3 on your keyboard or click the icon in the top left corner.



Now press E key to activate the extrude tool. Drag face until it touches the other pillar.

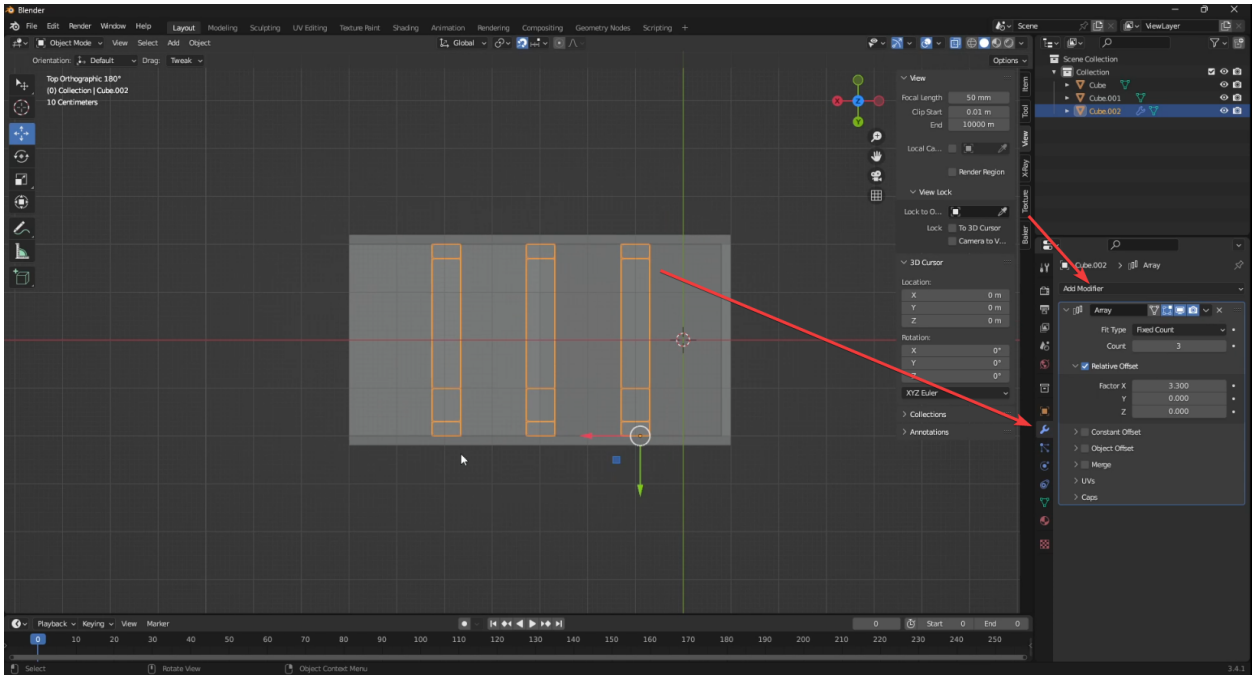


For optimization reasons let's delete internal (non visible) faces. Select them and press X



Now let's add more pillars like that. Go back to Object mode (TAB key). Do not copy objects with CTRL +C, CTRL + V. This will copy entries mesh, materials and other properties. In most cases we don't need that. Use Shift +D instead. Since we want to have the same pillars in a few places in the map I suggest you use a linked copy. Press ALT + D and place a copy somewhere. Now if you edit any of the copies the modification will be applied for each linked object on the map.

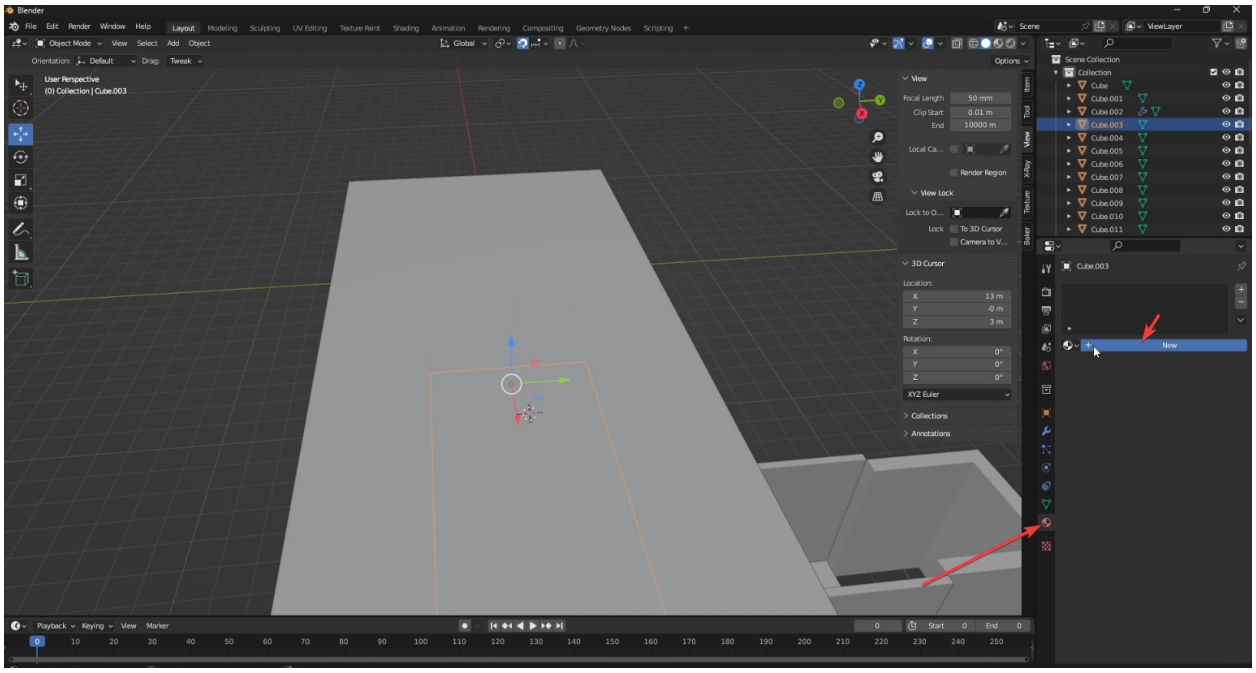
Another cool feature is the Array modifier. To add this modifier to an object click on the wrench icon in the bottom right corner. You can easily select how many times you want to repeat objects and offset them.



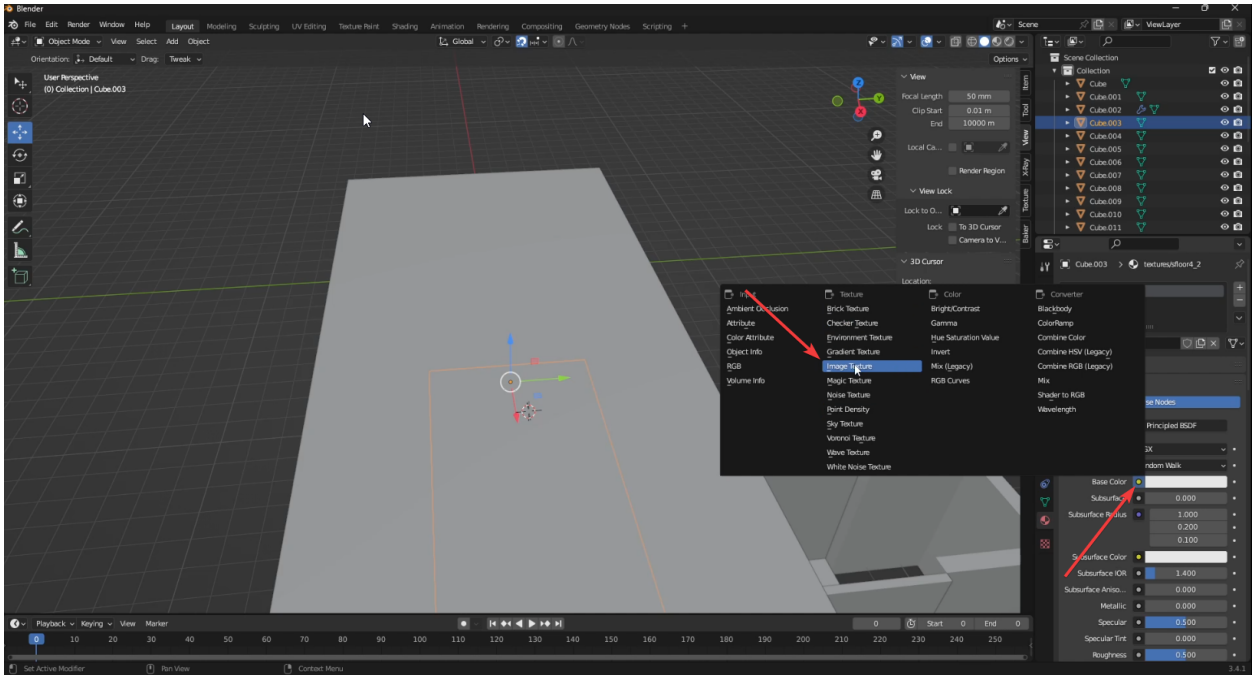
That is a basic tutorial for adding geometry on your map. This knowledge is enough to create a basic level. For further information check other blender tutorials you can find on the internet.

Texturing

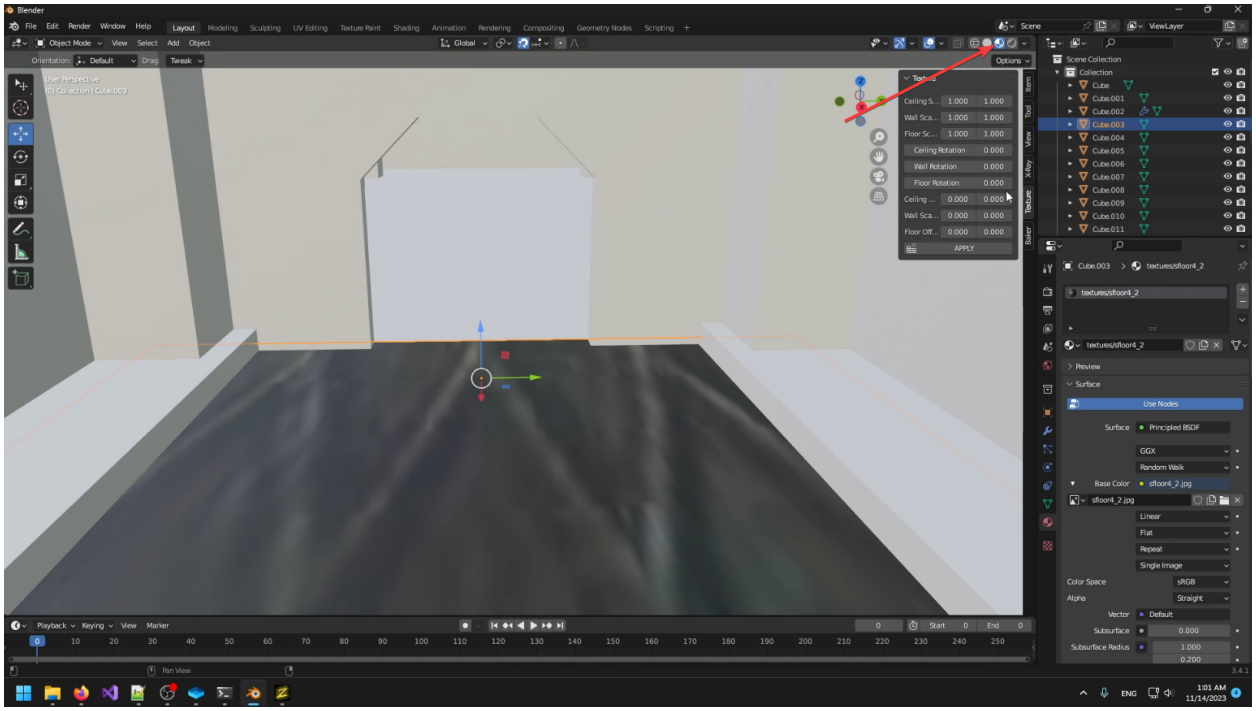
You may note our level is missing something. Yeah, textures. Add new material by clicking on the red sphere in the top bottom corner and pressing New. Let's set a name of the material corresponding to our texture path (for example textures/floor01). Don't forget to copy your textures inside your game folder (quake/i01/textures/).



Now let's set the texture of our new material. Click on a small circle near the Base color property and select Image Texture.



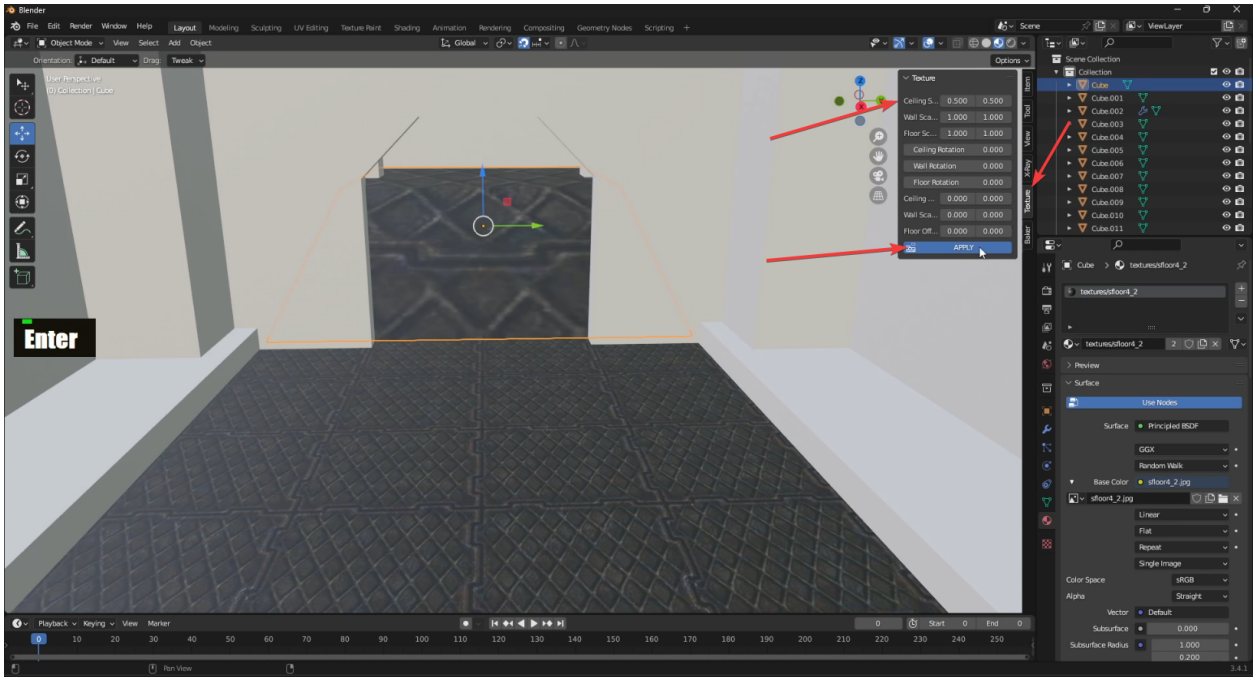
You may not see any changes. That is because your current shading mode is set to Solid. Click on another sphere in the right top corner to switch view to Material Shading. After that you may see that our texture is stretched and ugly.



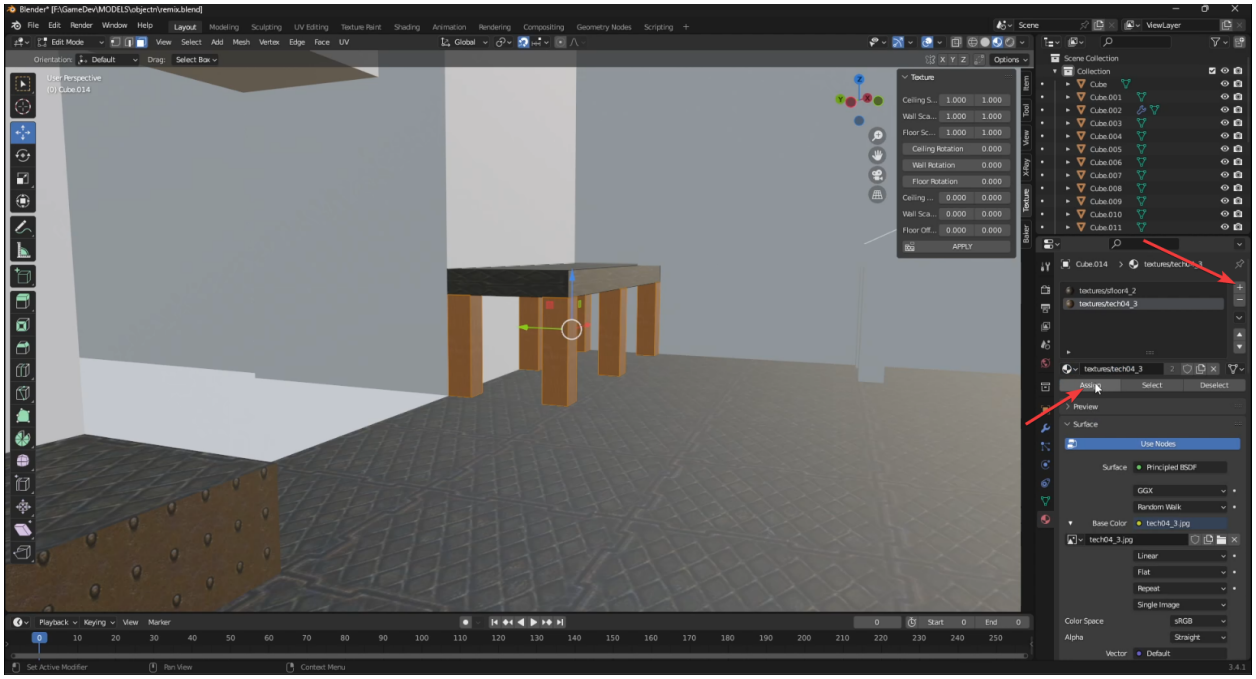
You can edit UV in UV Editing tab but for this tutorial I will show you a quick and easy tool that will tile texture like it is trenchbroom. Visit my addon page on github (https://github.com/KleskBY/KleskBY-s-Blender-Tools/tree/main) and download addons (Click Code > Download ZIP).
To install addon you can open Blender Preferences (Edit > Preferences), select Addons tab and press Install. Select the ZIP folder. Or just copy all .py files inside the Blender addons folder (C:\TOOLS\Blender 3.4\3.4\scripts\addons). Make sure they are enabled in the addon list.

For texturing we will need **TextureFill.py**
This addon allows you to easily assign and tile textures over objects. Works like Trenchbroom or other BSP tree software.

After you enable this addon you should see the Texture panel appear at the right. Click apply and see what is changed. You will see our texture perfectly tiled. To change texture scale edit the first 3 properties. I recommend values like 0.5. Click Apply to see the result.



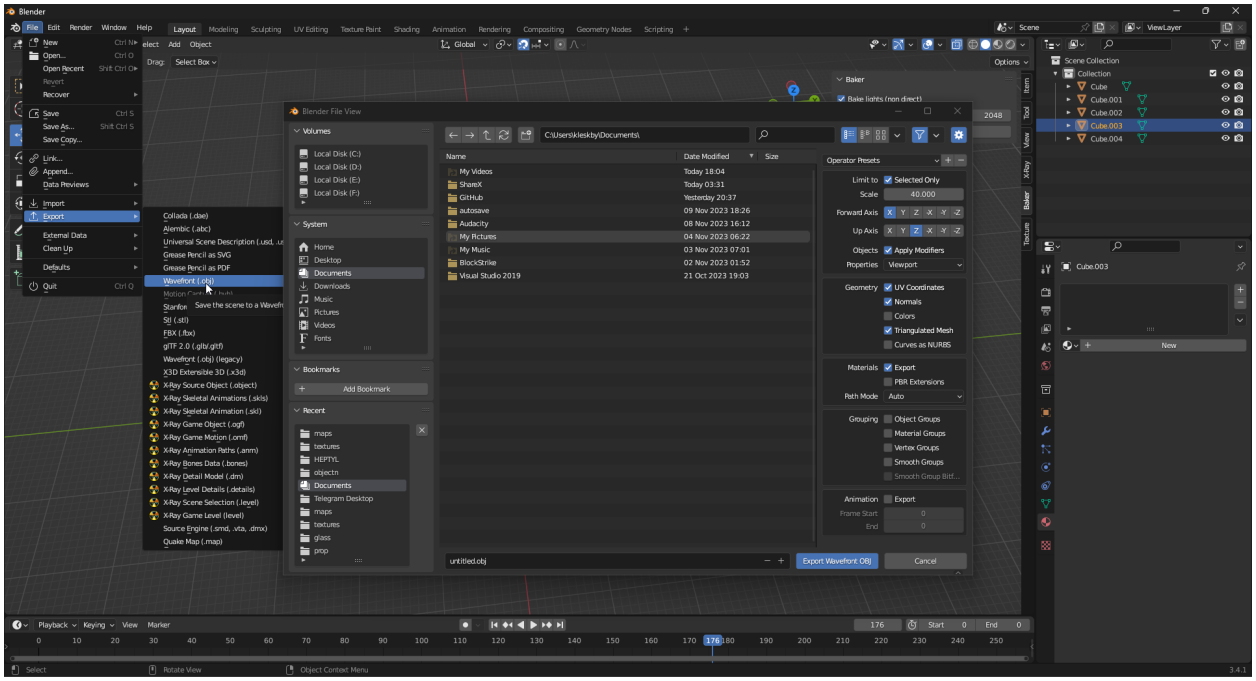
For some objects we will need multitexturing (few materials on the same object). Click the plus icon to add another material. Select a material. Now in edit mode select faces you want to assign another material too. Click the assign button.



That was a base texturing tutorial but you still need to learn about procedural materials and making a proper UV inside blender without 3rd party addons.

Export

Now we do have some geometry and textures. It is time to load our level in the game.
Press File > Export > Wavefront (.OBJ)
Since Blender uses metes (Metric system) and Quake uses feets (Imperial system) we need to adjust scale. Set the scale to 40. Enable Triangulate Mesh. If you want to use mod_obj_orientation 0 set Up Axis to Z and Forward to X otherwise leave as is. Export file to maps folder (quake/id1/maps).



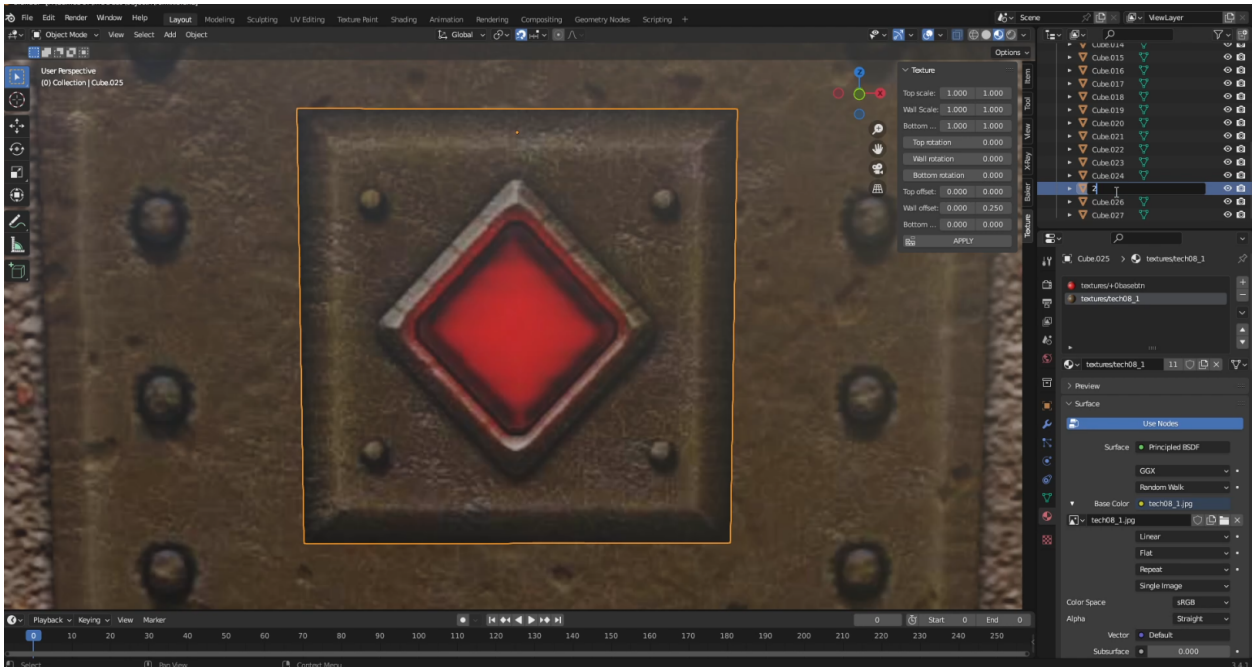
Now lets create a text file inside the maps folder. It should be named with your map name and have .ent extension. So in my case if a map is named untitled.obj I need to create a file untitled.ent.
Inside this file we put some basic information about entities such as worldspawn and info_player_start. Without it the game will crash since it can not find player spawn points. Put this inside your file.

```
{
"classname" "worldspawn"
"_fog" ".3 .35 .35 .42 .94 10 16384 1024"
}
{
"origin" "0 0 64"
"classname" "info_player_start"
"angle" "-90"
}
```

Now load the game, type “sv_cheats 1” (just in case we need noclip), and “map untitled.obj”. After that type r_fullbright 1 or r_ambient 20 since we have no lighting yet.

Entities

Hey, our map is still missing something. Doors, Elevators, enemies and more. Yeah, let's add an elevator to our map. Create an object that will be our elevator. Name it 1 (it is very important). Now add some kind of button. Name it 2.



Export your map again. Now in your .ent file add the following:

```
{
"classname" "func_door"
"model" "*1"
"lip" "1000"
"angle" "-1"
"targetname" "elevator"
}
{
"classname" "func_button"
"model" "*2"
"target" "elevator"
}
```

Got it? It is easy. Now our map has a button and an elevator that are connected together by some logic. Restart level to see changes (you may need to type r_restart or restart game). Let’s add enemies. Load map and type “r_editlights 1” in the console. You will see a cursor. In the right corner you will see the exact location of that cursor. To add monster simply add:

```
{
"classname" "monster_army"
"origin" "368 -740 24"
}
```

Lighting

Q1 Compilers build lightmaps and keep them inside BSP. We don't have this, but we have a few workarounds:

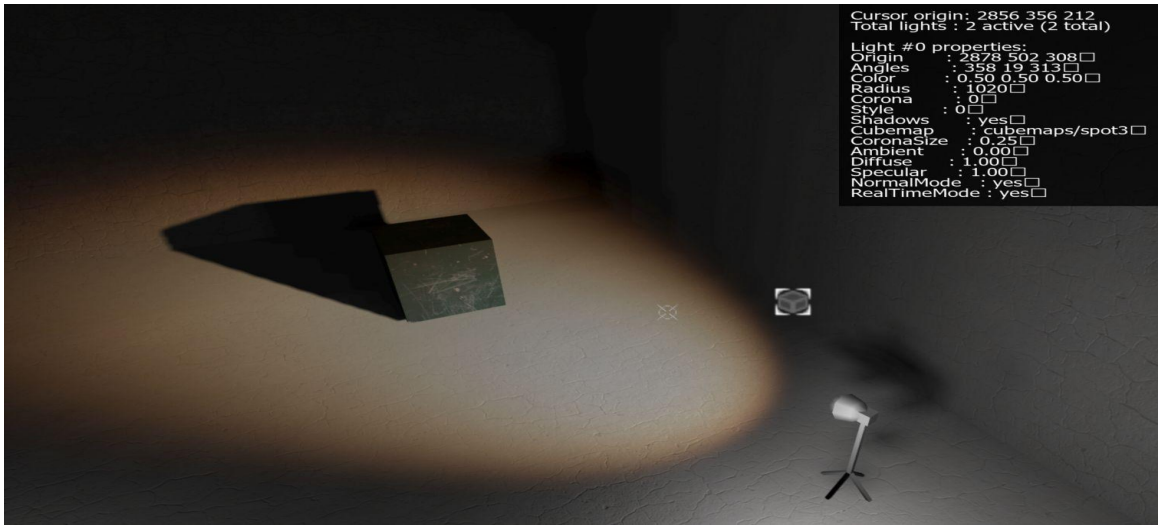
- Force full dynamic lighting (rtlights). You may just force r_shadow_realtime_world to be always true but this is gpu consuming and has performance issues with many lights on.
- You may use the “mod_generatelightmaps” command to bake rtlights. This is very buggy and unstable but may work for you.
- Bake lightmaps in Blender. This is the most optimal solution. Blender has more light features. You can achieve interesting and beautiful results.
- No lighting. Depending on your game style you may not use any light sources. For example GTA:SA has no lights.

Dynamic lights (rtlights)

Real-time lighting allows for more immersive and dynamic lighting effects within the game world. This includes dynamic shadows, reflections, ambient occlusion, and other lighting phenomena that respond to changes in the environment or in-game events. To add rtlights to your map lets first add a few helper commands to autoexec.cfg (create it in the id1 folder).

```
//EDIT LIGHTS
alias "editlights" "editlights1"
alias "editlights1" "r_editlights 1; alias editlights editlights2"
alias "editlights2" "r_editlights 0; alias editlights editlights1"
bind KP_INS "editlights"
bind KP_LEFTARROW "r_editlights_edit move -1 0 0";
bind KP_RIGHTARROW "r_editlights_edit move 1 0 0";
bind KP_UPARROW "r_editlights_edit move 0 1 0";
bind KP_DOWNARROW "r_editlights_edit move 0 0 -1";
bind KP_UPARROW "r_editlights_edit move 0 0 1";
bind KP_HOME "r_editlights_edit move 0 -1 0";
bind KP_PGUP "r_editlights_edit move 0 1 0";
bind KP_5 "r_editlights_spawn;"
bind KP_DEL "r_editlights_remove;"
bind KP_ENTER "r_editlights_save; echo SAVED!";
```

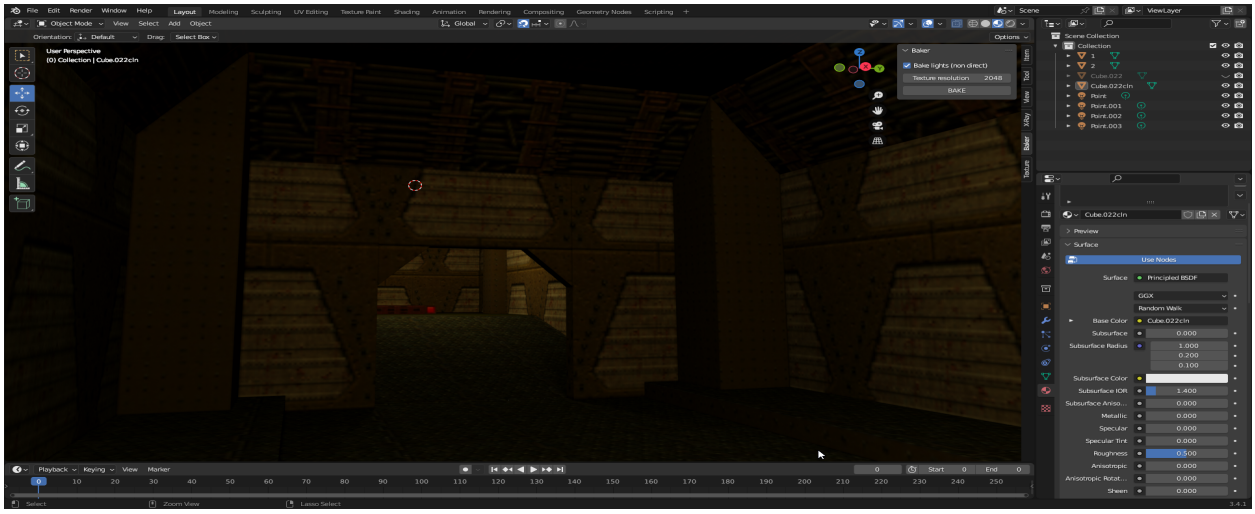
Restart the game or type “exec autoexec.cfg” in the console. Make sure real time lighting is enabled (r_shadow_realtime_world 1). From now, if you hit numpad insert (0), you will enable editlights mode (r_editlights 1). To spawn a light hit numpad 5 (r_editlights_spawn). To move light use numpad arrows. You can type r_editlights_edit origin <x, y, z> to set the position of the light. Similar for editing other properties.

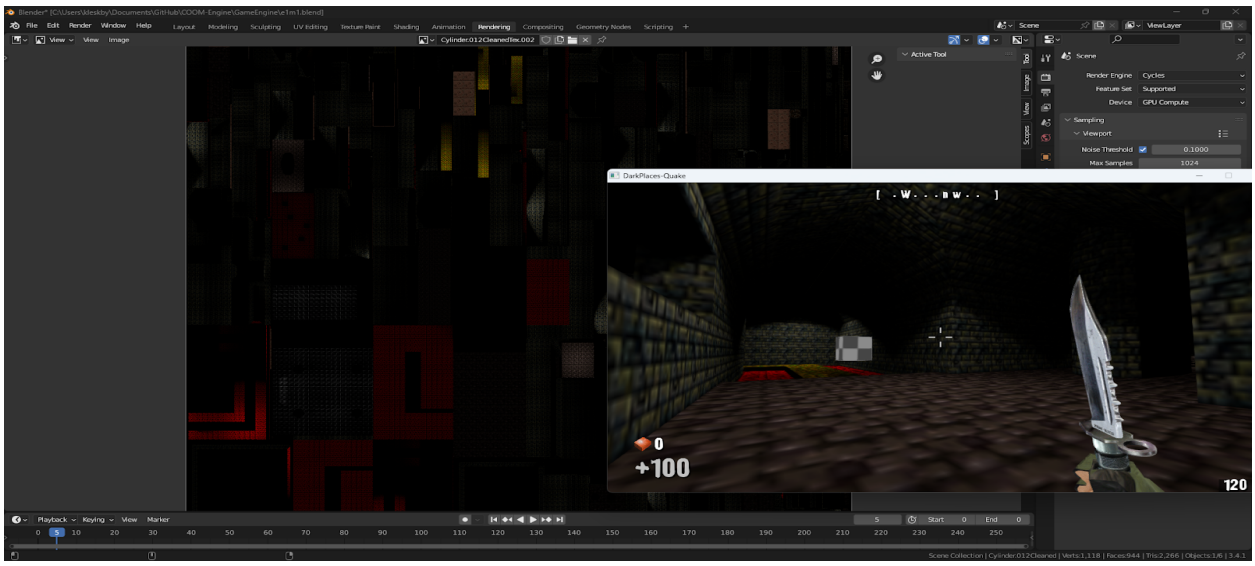


Baking textures and lights in Blender

BakeLightmapsToTexture.py

This addon is a magic button that will automatically bake textures and/or lightmaps to a single image to export or use inside the game engine. Enable this addon like you did in the Texturing chapter. Now you should see the Baker panel. Combine your world mesh (CTRL + J) into one/few objects.Switch shading mode to “Rendered”. Now hold SHIFT and click into the place where you want to put your light. This will move the cursor (red and white circle) to the desired place. Now press Add > Light >Point. Edit position, power, color as you like. Now go to the Baker tab, check “Bake lights” checkbox if you want to bake lights to texture and click Bake. After some time you will see a copy of your level with new material applied to it. If it is too pixelated, increase the resolution in the Baker tab.
To export the newly generated texture click on the Rendering tab. Select image. Click Image > Save As. And save it inside the textures folder as a Targa file. Edit the material name to make sure it is correct.





Occulition

Unlike BSP, OBJ has no VIS data. This means every polygon, light, entity is visible to the player no matter what direction he is looking. This creates serious performance issues on large maps. I think occlusion culling was not implemented by LadyHavoc since no one really was interested. Most maps will give you very good performance without any culling as long as you don't use lots of rtlights, but I came up with this nasty solution. Make each room in your map a separate object. In serverqc create culling.qc. Add cullin.qc to your progs.src. Add this code to culling.qc:

```
float CULL_DIST = 1500;
.float lastCull;
float isVectorInsideBoundingBox(vector vec, vector min1, vector max2)
{
    if (vec_x < min1_x || vec_x > max2_x) return FALSE;  // Vector is
outside the bounding box along x-axis
    if (vec_y < min1_y || vec_y > max2_y) return FALSE;  // Vector is
outside the bounding box along y-axis
    if (vec_z < min1_z || vec_z > max2_z) return FALSE;  // Vector is
outside the bounding box along z-axis
    return TRUE;  // Vector is inside the bounding box along all
dimensions
}


float() cullent_think =
{
    if(self.lastCull + 0.05 > time) return TRUE;
    self.lastCull = time;

    if(isVectorInsideBoundingBox(other.origin, self.absmin, self.absmax))
    {
        self.effects = 0;
    }
    else
    {
        local vector center = (self.absmin + self.absmax) * 0.5;

        makevectors(other.angles);
        vector vec = normalize(center - other.origin);
        float dot = vec * v_forward;
```

```
        if(dot > 0.25)
        {
            float dist = vlen(center - other.origin);
            if(dist < CULL_DIST) self.effects = 0;
            else self.effects = EF_NODRAW;
        }
        else self.effects = EF_NODRAW;
    }
    return TRUE;
};


void() func_wall_culled =
{
    self.angles = '0 0 0';
    self.movetype = MOVETYPE_PUSH;
    self.solid = SOLID_BSP;
    setmodel(self, self.model);
    self.customizeentityforclient = cullent_think;
};
```

This code defines a new type of entity - func_wall_culled. Next it checks if the object is in front of the player and checks distance to the player. If it is not in the front or too far away, not drawing this entity.

Another version of culling algorithm:

```
.string visible1, visible2, visible3, visible4, visible5, visible6;
float() cullent_think =
{
    if(self.lastCull + 0.05 > time) return TRUE;
    self.lastCull = time;

    traceline(other.origin, other.origin - '0 0 64', TRUE, other);
    if (trace_ent == self)
    {
        self.effects = 0;
        local entity head = find(world, classname, CULLING_CLASS);
        while (head)
        {
            if(head == self ||
            head.model == self.visible1 || head.model == self.visible2 ||
head.model == self.visible3
            || head.model == self.visible4 || head.model == self.visible5)
            {
                head.effects = 0;
            }
            else head.effects = EF_NODRAW;


            head = find(head, classname, CULLING_CLASS);
        }
    }
    return TRUE;
};


void() func_wall_culled =
{
    self.angles = '0 0 0';
    self.movetype = MOVETYPE_PUSH;
    self.solid = SOLID_BSP;
    setmodel(self, self.model);
    self.customizeentityforclient = cullent_think;
};
```

This version requires you to define visible chunks for each chunk. Here is how it looks like in .ent file:

```
{
  "classname" "func_wall_culled"
  "model" "*1"
  "visible1" "*28"
  "visible2" "*19"
  "visible3" "*33"
  "visible4" ""
}
{
  "classname" "func_wall_culled"
  "model" "*2"
  "visible1" "*4"
  "visible2" "*6"
  "visible3" ""
  "visible4" ""
}
{
  "classname" "func_wall_culled"
  "model" "*3"
  "visible1" "*4"
  "visible2" ""
  "visible3" ""
  "visible4" ""
}
{
  "classname" "func_wall_culled"
  "model" "*4"
  "visible1" "*2"
  "visible2" "*3"
  "visible3" "*5"
  "visible4" ""
}
```

MAP (BSP) Export

There is an addon that allows exporting Blender scenes to idTech .map file format that you can later compile. See the full page for details: https://github.com/c-d-a/io_export_qmap

I recommend setting the scale to 40 and mode Brush. But before that separate all unlinked meshes to avoid issues.

Other issues

TODO