Engine22 mapping - Sector Modelling

1. Background

Please read the "E22 Mapping – World Editor" document first to get a good understanding of the workflow and "why's".

2. What is a "Sector"?

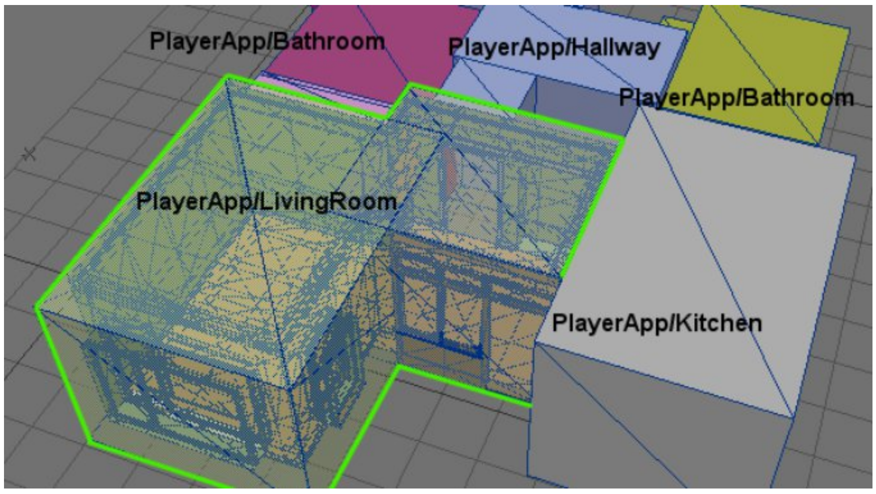
E22 Worlds (= the entire world, or a complete "level") are split into smaller "Sectors". Sectors are typically:

- A room
- A corridor (or corridor section in case it's a long one)
- A X square meters piece of ground for outdoor worlds

In the design phase, designers will split up the world in sectors so the artist that works out the actual geometry/textures/props/…/ doesn't have to care. Nevertheless, some basic knowledge is valuable;

Sectors are tight with the "Portal Culling" (http://tower22.blogspot.nl/2011/07/x-ray.html) technique that can be used in Engine22 to only draw what is necessary; the player stands inside a sector, and may see neighbour (or far background) sectors via "Portals", which are usually doors, windows, holes or just open-spaces.

Each sector will get its own *.e22sec file. Depending on your position in the world, the surrounding sector-files will be loaded. Which sectors these exactly are, is defined in the "World Editor" by the designers, and usually depends on factors such as your viewing range. An open world will typically load a lot of (low detail) sectors, while an indoor game like Tower22 can limit to closer rooms only.



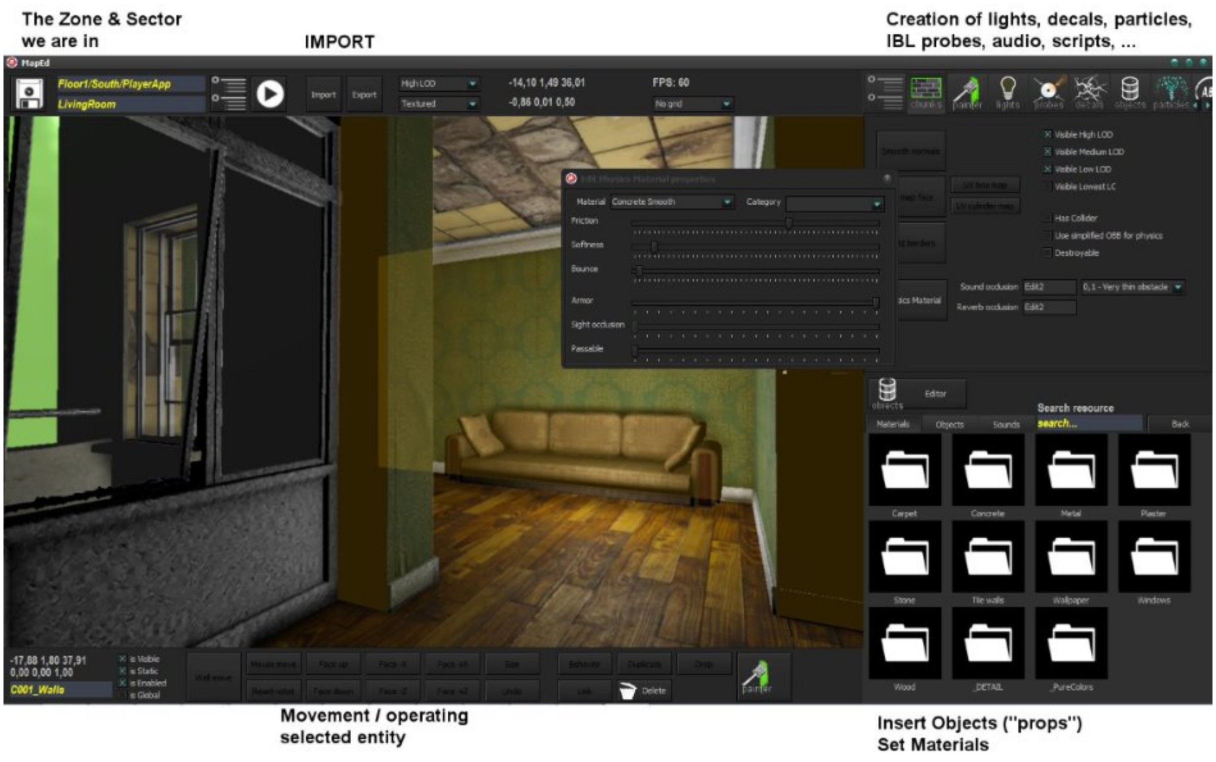
This apartment has been split up into 5 sectors.

Entities (monsters, barrels, items, furniture, lights, audio-emitters, particles, decals, …) will be linked to a parent-sector when they get inserted. When an entity moves, it can eventually be moved over to another sector when it crosses the borders (which is why we have to define global boundaries, to quickly sort this out).

Local entities are saved within a *.e22Sec file. Global entities that tend to move around or might be added/removed on a certain event, are saved in a separate (global game-state) file. The player, monsters, collectable items, puzzle related objects or destroyable stuff are examples of that. Local entities on the other hand will return to their original positions/state when the player left a sector (far enough) and returns. The magical clean-up service.

3. Editors / MapEd

Engine22 provides the "MapEd" (Map Editor) program for importing geometry, decorating (= putting entities), altering material parameters, creating objects, scripting, and so on.



However, it does not provide actual mesh-modelling though, asides from some basic operations such as:

- UV scale / shift / auto (box projection) mapping
- Changing materials per "chunk"
- Chunk movement
- Vertex movement
- Vertex welding
- Vertex (weight) painting for multi-layered materials ("Entropy shaders")
- Normal / Tangent calculation & smoothing

So you can alter the geometry a bit, but the actual creation shall be done in an external modeller program such as Lightwave, 3D Max, Blender or Maya. The reason why?

- Let the artist keep using his favourite tools
- Limited programming resources – cannot program a full features 3D modeller "quickly"
- And even so, the programs mentioned above are miles ahead with years of experience and thousands/millions of users. Just can't beat that.

That means we'll be importing geometry (= triangles, vertices, UV coordinates, optionally material assignments, …). So far MapEd supports LWO (Lightwave) and OBJ file formats. The latter is simple, and correctly implemented in almost any 3D modelling program. But possibly some more formats will follow in the future.

However, since OBJ is a limited format, we can only export limited data as well. Therefore we made some rules, which is what this document is really about.

4. Modelling rules

When saving an OBJ (or LWO) file ready to get imported into MapEd, thou shall

- **Triangulate** your geometry
  - No 2-point lines, 4-point quads, or n-point polygons
  - Avoid "bad" geometry (2 vertices at the same location will generate an infinite thin triangle)

- Avoid advanced types
  - Lights, Skeletons, Joints, Shaders, Physics, Camera's, Scripts, …
  - You can have them in your file, but they will get rejected in the OBJ export anyway. It's really about geometry and "naming" only

- Decide what is going to be "**Static Geometry**", and what can be a (recyclable) "**Prop**"
  - Static stuff is usually walls, floors, stairs, pipes, trims, beams, …
  - Stuff that can be tossed / altered / kicked (use physics) = prop
  - Stuff that can be picked up (items) = prop
  - Stuff you will likely re-use (lights, furniture, junk, devices, crates, …) = prop
  - Stuff that can move/think/animate = prop
  - Stuff that can be broken = prop…possibly…discussion
  - Doors (the rotatable/moving part at least) = prop…possibly…discussion
  - **Props do not belong in the Static Geometry**
  - We'll produce "Objects" for props, and place them later on. But that's another story.

- **Tessellate** (sub-divide into smaller quads/triangles). IF needed.
  - Tesselation will generate more vertices
  - And vertices can be drawn. Which is very useful for multi-layered materials, or custom shading your worlds ("baking light manually")
  - And they can also reduce normal smoothing artefacts
  - But do not end up with thousands of polygons for a simple wall please!
  - The average Tower22 room is between 400 and 4.000 triangles. Larger area's will consume more of course, but just to get an idea.

- Optimize
  - Certainly if you expect lots of stuff on the screen / distant views
  - If you can't see it, don't model it (mainly talking about back-sides).

- **Weld** vertices!
  - Avoid tiny gaps between faces. It's #@$ ugly.

- Respect the **boundaries**
  - See "World Editor" document. Triangles that stick outside the boundaries shall be cut off.
  - If boundaries are incorrect, kick the designer. He should adjust the boundaries then.

- **Apply the right names** on your triangles (groups)
  - This will tell the importer what the hell this triangle is about. See next chapters.

5. Chunks

Since the OBJ file format is kinda primitive, and we'd need to import advanced stuff, we need to give a helping hand. This will cost some time during the modelling process, but it will pay off.

Usually we can assign a material to each triangle. This material has 2 properties we're interested in:

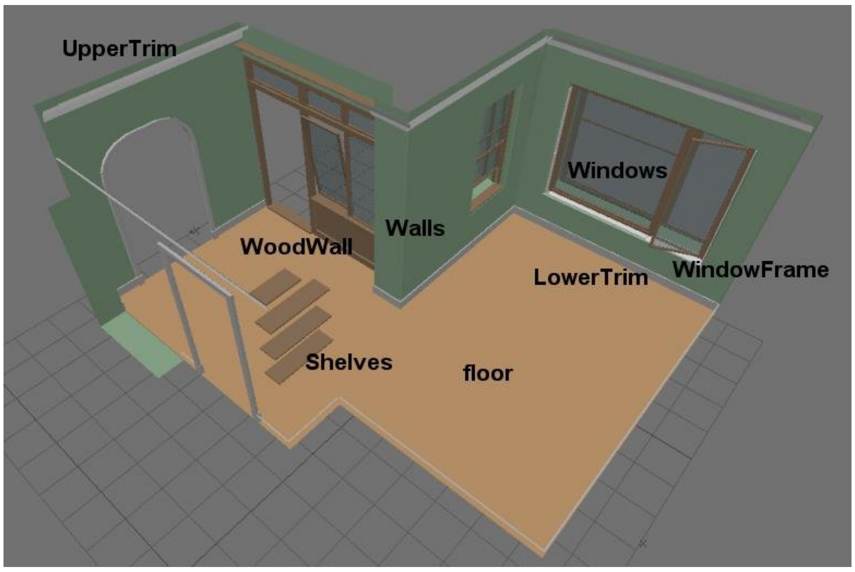
• A name	→ So we know what this triangle is, and where it belongs to
• A texture ("brickwall.tga")	→ So we can automatically try to find our corresponding materials

When doing maps, the most common type of grouping you'll do, is separating your sector into smaller chunks. Think about "left wall", "right wall", "floor", "pillar1", "stair railing", "window4".

Once imported In MapEd, we don't select per triangle, but per "chunk". And each chunk has:

idName	So we can identify in the editor, as well search & alter it in-game via scripts ("wall1.show")
Material	Brick wall, carpet floor, … texture(s) + shader + parameters
Position/rotation	We said "static", but chunks can actually move. Think about moving platforms, a bridge going up, …
Visible yes/no	Chunks can be shown/hidden on command. For example, we could something with a destroyed variant.
Custom behaviour	To drive movement and such.
Physics collision shape	As the name says, the "hull". If any, maybe we want to alk right through it.
Physics properties	Friction, Elasticity, Softness, kind of impact sounds / particles / decals when bullets or objects hit
Audio occlusion properties	The amount of reduction when noise was emitted on the other side of this chunk.
Level of Detail info	Still visible after X meters? Small / less important chunks can be hidden for lower LOD (distant) sectors, so we can create larger open worlds without having to draw billions of barely visible triangles.

Yeah, I bet you didn't think about all of that while doing the 3D model. In the end, your 3D model will get a lot more "game-related" properties. So it makes good sense to group your triangles into chunks.



Equal colors belong to the same chunk here. Note that the triangles do not have to connect. In this example, all 3 windows belong to the same "Windows" chunk. So we can edit them all at once. In fact, less chunks → better performance. But of course, since you can only have 1 material & set of properties per chunk, you may want to split up further anyway. Your choice.

6. Triangle Naming

So how does the “chunk-grouping” work? By naming their assigned material:

C<chunkID>.<idName>
C1.LeftWall
C1.RightWall

Where “C” stands for Chunk (note we can also model other non-visible stuff into the same file! Therefore the indicator). As for the ID number, this is a bit more complicated. Initially, we don’t care (so just name it 1 or something). When importing for the very first time, Engine22 will create new chunks, and generates an absolute unique “entityID” for each – a long 64 bit number.

Export & Re-import
Now it may happen we need to make some geometry adjustments, but already did modifications in MapEd as well. We don’t want to restart, so instead we export the e22sec file back to an OBJ file (via MapEd). Now the C1 gets replaced with an actual id number, something like C423620214.Rightwall.

If we now edit “RightWall”, and re-import into MapEd again, Engine22 will recognize this chunk. Instead of creating a new one, it will replace the mesh of chunk 423620214.

Modelling in your favourite editor is great, but a big flaw is that you’ll have to ping-pong files back- and forward between Blender/Max/Maya/Whatever and MapEd. Now at least we can keep as much as possible intact by using these idNumbers, instead of having to restart every time.

Besides “C”(hunks), we have a few other types of geometry, though this part is still under construction as we write, so don’t try this at home yet:

C	C1.LeftWall	Chunk. See previous chapter.
R	R1.Sewerpipe	Audio reverb. Applies a local reverb effect within a zone, modelled as a sphere.
P	P1.Water	A (pointcloud) volume wherein we apply a certain physics effect, such as water buoyancy or anti-gravity
D	D1.idName	Decal
L	L1.LightType	If we like to, we can model lights inside our modeller as well – instead of adding them afterwards via MapEd
X	X1.idName	Particle volume

7. Materials & UV Mapping

Besides a name, materials usually also have a texture. “Grass1.bmp”, “brickwall.tga”, “rustypipe.dds”, et cetera. We don’t have to, but it sure helps to assign textures during the modelling phase already. If we have them available of course, otherwise a dummy could be used as well.

Anyhow, if “stinkyPipe01A.png” was applied, MapEd will look for a material called “stinkyPipe01A” (or a close match without the 01A for example) and automatically assign it. Which saves another step of picking chunks and assigning materials. But if it didn’t find anything, a default material will be selected.

Since we match on image filenames here, it’s a good idea to name your material file in the same fashion as the (diffuse/base) texture it uses.

Also, for working, all textures are stored in the <yourgame>/Data/Materials/…sub… folders. So while modelling, its probably handy to keep this same folder at hand to get a “database” of available textures so far.

Note that multiple chunks can refer to the same image, and thus use the same material.

With the right (dummy) textures, we could just as well UV-map the thing as well in our modeller, don’t we? MapEd has some UV mapping tools (shift, scale, auto-map by planar/box/sphere or cylinder projection). The advantage of them is that they can be used very easily to quickly generate UV coordinates. Then again the tools are kinda crude and only work well for simple box-like shapes. Bended pipes, organic blobs, curled carpets or whatsoever are better off with an advanced UV mapping set.

If UV-Mapping is a painfully slow process, you might want to import without UV coordinates into MapEd first, let if generate auto-UV maps, export back to OBJ, and then refine the UV maps in your editor.