Building Optimal Maps

- Introduction: the need for efficiency
- Optimal use of materials: case study
- Effective use of sectors: case study
- Dynavis visibility culler: case study
- How to use renderpriorities
- Binary alpha for materials
- Faster loading
- Using regions
- CEL zone manager

Optimal Maps: the need for efficiency

- Hardware gets better and better. So why not just make one big map containing everything?
  - As hardware capabilities rise, user expectations also rise!
  - Need more geometry, more detail, more effects, ...
  - Worlds are getting more complicated
- Additionally support for old hardware is sometimes still desired

Optimal Maps: optimal use of materials 1

- Better 10 objects with 10000 triangles each then 100 objects with 1000 triangles each
- Number of distinct objects is biggest cause of slowdown:
  - 3D hardware prefers fewer objects (batches)
  - Visibility culler and engine in CS prefer fewer objects
  - Collision detection system prefers fewer objects
- With extensions like VBO (ARB_vertex_buffer_object) number of polygons is less important
- Note! One mesh object for engine == potentially multiple objects for renderer!

Optimal Maps: optimal use of materials 2

- WARNING: bad programmers art ahead!
- Showcase: 165 houses (17395 faces), each house uses 4 materials:
  - Case 1: 165 meshes, 4 materials per mesh: 31 FPS
  - Case 2: 165 meshes, 1 material per mesh: 56 FPS
  - Case 3: 16 meshes, 4 materials per mesh: 55 FPS
  - Case 4: 18 meshes, 1 material per mesh: 62 FPS

Optimal Maps: optimal use of materials 3

- Combining materials is good. But:
  - Can be hard because of tiling. Possible solutions: pre-tiling or splitting triangles
  - Finding efficient combined materials is not always easy and possible
  - This is harder for the artist
- Combining objects is good. But:
  - Again harder for the artist
  - Memory usage is worse because factories cannot be shared easily

Optimal Maps: effective use of sectors 1

- Advantages of sectors:
  - Cleanly separate logical areas in a map (for example indoor versus outdoor and different rooms)
  - Possibilities to have different artists work on different sections of a world easily
  - Facilitates dynamic loading of worlds
  - Allows for special effects (warping portals)
- Disadvantages:
  - Game logic may be harder for cross-sector computations
  - Somewhat harder for artists to use effectively

Optimal Maps: effective use of sectors 2

- Showcase: 401 objects (90151 faces):
  - Case 1: one sector only: 15 FPS
  - Case 2: seven sectors: 36 FPS
  - Case 3: one sector with Dynavis: 30 FPS

Optimal Maps: effective use of sectors 3

- Sectorizing can help performance considerably
- Dynavis is almost as good
- PVSVis is expected to be better (waiting on GSoC!)

Optimal Maps: Dynavis visibility culler 1

- Default visibility culler in Crystal Space is frustvis:
  - Only does frustum culling
  - Based on hierarchical kdtree for objects
- Alternative is Dynavis:
  - Highly suitable for dynamic worlds
  - Tries to see when objects are occluded by other objects
  - High overhead on CPU

Optimal Maps: Dynavis visibility culler 2

- Showcase: 545 objects (65452 faces):
  - Case 1: default frustvis culler: 17 FPS
  - Case 2: dynavis culler: 218 FPS
- Note: switching culler at runtime using BugPlug:
  - Ctrl-d ctrl-shift-1: frustvis
  - Ctrl-d ctrl-shift-2: dynavis

Optimal Maps: Dynavis visibility culler 3

- Conclusion:
  - Dynavis can help a LOT in some cases
  - Especially suitable in maps where sectorizing is not easy
  - PVSVis will be better still!

Optimal Maps: renderpriorities 1

- Objects in a sector are rendered in order of render priority (so first 'init' and then 'sky' and so on
- There are 11 predefined priorities (init, sky, sky2, portal, wall, wall2, object, object2, transp, alpha, final):
  - 'sky' and 'sky2': used for skybox/skydome (znone/zfill)
  - 'portal': often used for special types of portals (warping)
  - 'wall': used for regular outer sector walls (zfill)
  - 'object': used for normal objects (and keycolor) (zuse)
  - 'transp': used for orderable transparent objects (ztest)
  - 'alpha': for other transparent objects (ztest)

Optimal Maps: renderpriorities 2

- Most maps will use 'object' render priority only
  - Requires z-buffer and screen to be cleared every frame
- 'alpha' render priority is special:
  - Objects in that render priority are sorted from back to front
- 'transp' render priority doesn't do that:
  - Only use for objects that are transparent but don't require sorting back-to-front (add mixmode, ...)
- Note: as soon as you have a single object that needs 'alpha' then all 'transp' objects have to go to alpha too

Optimal Maps: binary alpha

- Alpha transparency (alpha material or alpha mixmode) requires back to front sorting: 'alpha' render priority with 'ztest'
  - This is slow: sorting happens every frame
  - This often looks ugly: sorting is not perfect (only sorts on center point) so sorting errors occur (especially on big objects)
  - Sorting of object triangles is also required normally. Only genmesh supports that (using bsp tree): slower still
- Only 0% and 100% transparency is required: use binary alpha
  - Add <alpha> <binary/> </alpha> to the texture definition
  - No sorting is required. Use 'object' render priority with 'zfill/zuse'
  - Texture aliasing artifacts can occur

Optimal Maps: faster loading

- Using binary XML to speed up map loading (docconv tool)
- Using streaming loader. Not yet! Waiting for GSoC!
- Convert images to DDS. Precompressed and mipmaps already generated

Optimal Maps: using regions

- A region in Crystal Space is NOT a geometrical concept:
- It is a collection of Crystal Space objects (meshes, materials, textures, lights, sequences, ...)
- Regions don't have to correspond to geometrically close objects but they typically do
- Regions make it easier for the engine to manage a set of objects at once (i.e. unload/load)

Optimal Maps: CEL zone manager

- CEL has a zone manager which adds support for loading and unloading of regions
- Currently loading and unloading only occurs one region at a time
- No streaming yet (waiting on GSoC!)
- Concept of:
  - A map (i.e. A physical map as exported from Blender for example)
  - A region: a collection of maps that are treated as one unity
  - A zone: a collection of regions that are loaded together