



INTELLIGENT COMMUNICATIONS

web.alive | environment performance & optimization

October, 2010



Performance Dimensions



Download

Megabytes
MB/s
kilobytes
kb/s



Rendering

Frames/sec
FPS
Megatexels/sec
Polys/sec



Server

Updates/sec
Collisions/sec
PCU
kb/s

OPTIMIZING INITIAL DOWNLOAD



.wae file contents

- ▶ Unreal packages (Izma compressed)
 - .csm, .utx, .usx, .uax, etc.
 - *Unreal packages are downloaded in the white fog room*
- ▶ Insertions
 - .jpg, .png, .bik, .pdf, etc.
 - *Downloaded while exploring the environment*
- ▶ Meta data
 - *Not downloaded*

Example LZMA Compressed Sizes

Map	.csm	.utx	.usx	other	Total
Apartments	0.7	33.7	16.3	0	50.7
Bridges	0.5	7.3	1.9	0	9.7
Canvas	0.01	0.06	0.03	0	0.1
Discovery	0.8	6	2.2	0.5	9.5
Executive	0.2	3.2	0.4	0.2	4
Park	0.2	22.6	2.6	1	26.4
Training	0.5	9.9	1.5	0	11.9
Total	2.91	82.76	24.93	1.7	112.3
Percent	2.59%	73.70%	22.20%	1.51%	100.00%

Optimize your textures for the big win in download size

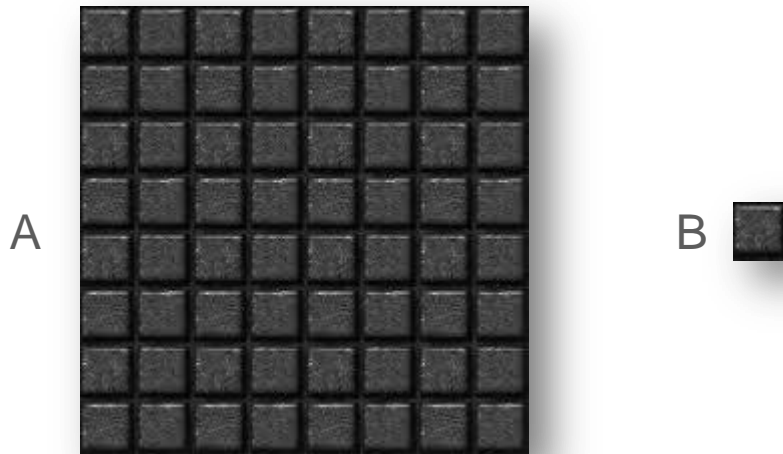
Caching and reuse

- ▶ Unreal packages are cached by web.alive (LRU)
- ▶ Unreal packages are identified by a unique identifier (GUID) and are only downloaded if a package with the unique identifier doesn't exist in the cache

You can share common packages across environments to further reduce download times

How to optimize textures (1)

- ▶ Select smaller texture dimensions
 - 1024x1024 pixel texture -> 4MB uncompressed
 - 256x256 pixel texture -> .25MB uncompressed



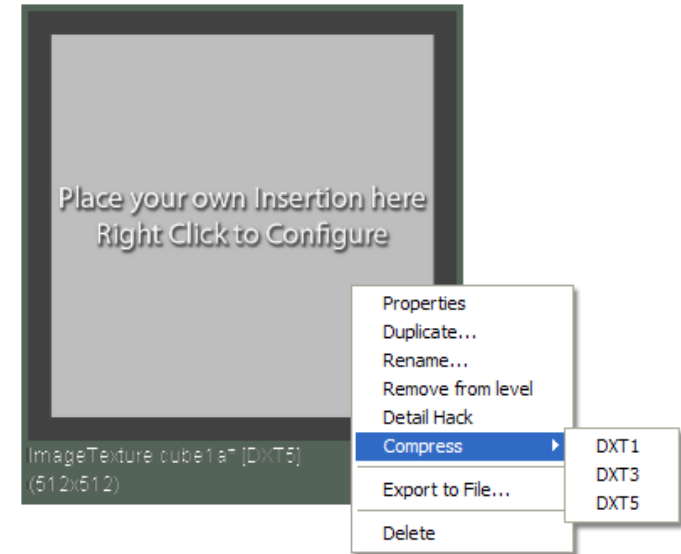
When tiled textures A & B produce exactly the same result

A note on texture size

- ▶ The default web.alive window is 900x400 pixels
- ▶ Under what conditions will the user get close enough to see more detail than a 512x512 pixel texture...

How to optimize textures (2)

- ▶ Compress your textures
- ▶ A 1024x1024 pixel texture is 4 MB uncompressed
 - 1.0 MB when DXT3 or DXT5 compressed
 - 0.5 MB when DXT1 compressed



DXT compressed textures are 4-8 times smaller

How to optimize textures (3)

- ▶ Use insertions
- ▶ A 1024x1024 pixel texture is 4 MB uncompressed
 - ~0 MB as an insertion
 - Pixel content is downloaded as a jpg in the background
 - 1024x1024 pixel texture compressed as a jpg may be <100KB

Insertions are 100% smaller on initial download and as much as 40x smaller on dynamic download

Tips & Tricks

- ▶ Download a copy of 7-zip so that you can open .wae files and view the compressed sizes of your packages
- ▶ Avoid including textures & texture packages that you aren't using in the .wae file
- ▶ Learn how to create tiled textures (see CDG)
- ▶ You can pack a lot of irregular textures into a single texture and use UV mapping to extract the right parts from the single texture (check out ut2004 or unreal runtime avatar textures for an example)
- ▶ A single texture can be modified by a shader to look very different and thus useful in many places (e.g. use a color modifier to change the color of your texture)
- ▶ Instead of using a single huge texture, you can often tile a small texture and then overlay a second texture for detail in a particular place (eg a brick wall with a logo in the middle). Overlays can be accomplished with projectors, combiners, or additional static meshes.

OPTIMIZING DYNAMIC DOWNLOADS



Dynamic downloads

- ▶ 3 content factors affect dynamic download speed
 - Insertion size (1024x1024 vs 512x512 etc)
 - Image type (.jpg vs .png)
 - Compression settings (.jpg / .gif compression options)

Smaller files result in faster dynamic image loading

OPTIMIZING RENDERING PERFORMANCE



Factors that drive frame rates

- ▶ Client GPU hardware
- ▶ # and type of polygons
- ▶ Size and complexity of materials
- ▶ Render steps
- ▶ Overdraw
- ▶ Total GPU memory consumption
- ▶ Anti-aliasing / Filtering etc.
- ▶ Particles, movers, scripts
- ▶ Competing demands (web rendering, voice, other apps)



Client GPU Hardware

- ▶ Minimum hardware for web.alive is the Intel GMA950 (part of the 945 chipset series)
 - 10MB/s memory bandwidth
 - 1600 megapixels / sec
 - No hardware transform or lighting (handled by cpu)
- ▶ Typical low end discrete graphics card: nvidia 9400GT
 - 25MB/s memory bandwidth
 - 2.2 gigapixels / sec
 - Hardware transform and lighting
- ▶ Typical discrete graphics is at least 3x faster

**You can have more complex environments
If you can control your client hardware**

Polygons

- ▶ web.alive supports three kinds of polygon objects: BSP, (static) mesh, and terrain
 - In general BSP should be kept simple as it is mostly handled in software (slow to render/triangle, but efficient collision)
 - In general complex shapes should be static meshes – mostly handled by the GPU (fast to render/triangle, but can be slow for collision). Static meshes are instanced, which means you have many copies of the same mesh at lower rendering cost.
 - Terrain should be used for large open spaces (medium to render/triangle, efficient collision)
- ▶ In general frame rates in web.alive are not limited by static mesh polygons (usually limited by other factors first), but may be affected by very detailed terrain

High poly static meshes are generally ok

Textures

- ▶ Textures consume a lot of GPU resources
 - Scaling textures is expensive, so MIP mapping was invented (at a cost of additional texture memory)
 - Texture memory is scarce, so DXT compression was invented (saves GPU memory and bandwidth transferring textures to the GPU)
- ▶ 256 MB video memory fits less than 50 uncompressed 1MP textures w/ MIPs (and that doesn't account for memory consumed by polygons, multiple buffers, shader programs, intermediate results, etc)

Too many large textures can cause frame rate hitches

Materials & Shaders

- ▶ Materials & Shaders bring an environment to life by making objects look glowing, wet, shiny, reflective, moving, etc.
- ▶ Materials and shaders work by combining and manipulating textures with various operations every time they are rendered
- ▶ A complex material may be the equivalent of rendering 10 or more traditional flat textures on top of each other (from a GPU load perspective)
 - ex. Water might be simulated with a 2 semi-transparent ripple textures, a panner (that moves one ripple texture), and a specular shader (which makes the water shiny and uses an environment map)
 - Because certain shader operations depend on the user's angle of view (specularity, reflections, etc) – they must be computed per pixel and not cached. This means that the visible size of such shaders in the current view will affect GPU load

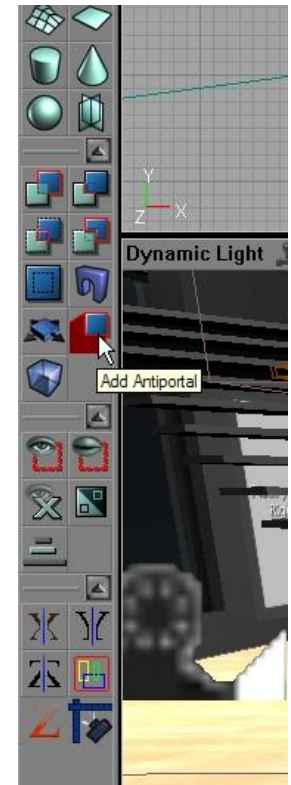
Shader complexity is often a frame rate limiting factor

renderer steps (sections)

- ▶ Unreal renders each mesh and each part of each mesh with a different material as a separate render step (section in unreal terminology)
- ▶ Each render step (sections) has a lot of overhead associated
 - so rendering 10 individual box meshes is much more expensive than rendering a single mesh of 10 boxes (even if the total polygons is the same)
 - Rendering a box with 6 individual textures is much more expensive than rendering a box with a single larger texture (even if the total number of texture pixels is the same)
- ▶ Stat render - unbatchedunsortedsections is a key performance indicator here (less than 400 is ok)
- ▶ <http://hourences.com/book/tutorialsmeshrendering.htm>

Overdraw

- ▶ How many times the same pixel is redrawn in a single frame
 - Modern GPUs have many tricks to reduce overdraw (hierarchical z, early z check, etc) -- not available on integrated graphics
 - Overdraw is inevitable with transparency (so reduce the opportunity to look through many transparent layers)
- ▶ Overdraw can be reduced by telling the engine when objects are completely hidden
 - via anti-portal occlusion (must be large simple shapes)
 - via BSP zones occlusion (best for enclosed areas – e.g. rooms)



Use anti-portals and BSP zones to improve frame rates

Anti-aliasing / Texture Filtering

- ▶ Anti-aliasing and texture filtering are technologies for improving the quality of 3D graphics, both increase the load on the GPU
 - 4x Full screen anti-aliasing is used by web.alive when anti-aliasing is checked (has roughly the same effect on GPU performance as doubling the window size in both dimensions)
 - Anisotropic/Tri/Bilinear filtering are methods for ensuring that scaled and projected textures remain sharp (adds roughly 5-25% to GPU load)

Anti-aliasing is expensive on low end hardware

Particles, movers, scripts

- ▶ Particle systems are great for simulating 3D moving / fluid things (fire, smoke, sparks, water splashing, etc)
 - Particle systems affect frame rate based on the number of particles, their size, particle collision and clipping
- ▶ Movers are generally low cost – however, if expected to cast shadows they may require additional meshes / projectors increasing scene complexity
- ▶ Scripts – not used in many of our maps, but complex/poorly optimized scripts can also affect frame rate. Try to avoid any complex logic in “tick” as it will have to be executed every frame.

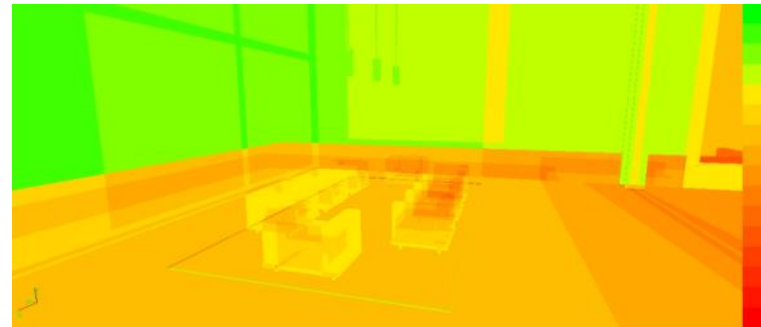
Competing demands

- ▶ Web rendering
 - Rendering a youtube video or flash animation can consume 30% of both CPU and GPU on low end machines.
 - Use activation volumes to control the number of web renderers (desktop sharing is actually web rendering a flash app) active at any one time
- ▶ Image/Document insertions
 - Image and document insertions consume cpu and network bandwidth during loading
 - Use activation volumes to delay activation of insertions until the user enters the volume
- ▶ Desktop sharing and other apps
 - Sharing your desktop and/or running other high CPU applications can also affect frame rates – not much a content developer can do except try to leave some head room.

Activation volumes are key to maintaining frame rates

Hints and tips

- ▶ Profile your environment using rmode 1, rmode 8 & stat all:
<http://udn.epicgames.com/Two/LevelOptimizationProfiling.html>
- ▶ Use anti-portals and zones to cull geometry that doesn't need to be rendered
- ▶ Break up large (in size not in polys) meshes so that parts can be occluded with antiportals and culled by zones and the view frustrum:
<http://hourences.com/book/tutorialszoning.htm>
- ▶ Use activation volumes to reduce CPU/GPU contention from web renderers and other insertions
- ▶ Try to stay within a max of 2 web renderers per volume
- ▶ Consider splitting extremely large / complex levels across servers (you can teleport between servers in 2.5 in a matter of a few seconds)



OPTIMIZING SERVER PERFORMANCE



Factors affecting server performance

- ▶ Relevance
 - Each actor that is “relevant” has its data synchronized across the network. Try to avoid “always relevant” actors.
- ▶ Map complexity / collision
 - The server is the master for avatar location and thus must compute collision for all avatars. Collision complexity greatly affects server scalability
 - Eliminate collision entirely where possible
 - Simplify collision meshes wherever possible (sometimes the best approach is to use a very simple invisible collision mesh in front of a complex set of meshes)



RESOURCES

Resources

- ▶ <http://avayalive.com/WaStore/Community.aspx>
- ▶ http://wiki.avayalive.com/projects/webalive/wiki/Content_Developer_Guide
- ▶ <http://udn.epicgames.com/Two/LevelOptimizationProfiling.html>
- ▶ <http://hourences.com/book/tutorialsindex.htm>
- ▶ http://wiki.beyondunreal.com/Legacy:Map_Optimization

