**Grayman’s notes on loading textures in parallel**

When looking at map loading, I identified three processes that consume a lot of time:

1. Loading and spawning entities
2. Compiling routing information (where the AI can travel)
3. Loading images

The most time-consuming of these is ‘loading images’. This takes up, on average, half the load time.

A list of required images is built while entities are spawned. No images are loaded at this time. I thought if I could use a separate thread to read the entities (w/o spawning them), build the image list, and start loading images, that would let process #3 run in parallel with processes #1 and #2. Process #1 would be changed to not build the image list, since that was now done by the new thread.

The parallelism starts at the point where the briefing video begins. This occurs when the player hits the “Start This Mission” button on the main menu page. This action is captured by idGameLocal::HandleMainMenuCommands() by this code section:

else if (cmd == "onstartmissionclicked")

{

// First mission to be started, reset index

m\_MissionManager->SetCurrentMissionIndex(0);

gui->SetStateInt("CurrentMission", 1);

ClearPersistentInfo();

\_thread = ThreadPtr(new boost::thread(boost::bind(&idGameLocal::LoadImages, this)));

}

The last line in that section starts a new thread that runs the idGameLocal::LoadImages() method.

Here’s the code for that method:

void idGameLocal::LoadImages()

{

// Get the starting map file name

const idStr& curStartingMap = m\_MissionManager->GetCurrentStartingMap();

if (curStartingMap.Length() == 0)

{

return;

}

m\_texturesLoaded = false;

idStr filename = va("maps/%s", curStartingMap.c\_str());

// Load the map from the missiondata class

idMapFile\* file = m\_MissionData->LoadMap(filename);

tdmDeclTDM\_MatInfo::precacheMap( file );

// Go through entities, looking for images.

PreprocessEntities(file);

// Load images

common->PreloadImages();

m\_texturesLoaded = true;

}

This basically takes care of two things:

1. PreprocessEntities() reads the entity data and creates the image list.
2. PreloadImages() loads the images.

I ran into two major problems:

1. The existing code is designed for single-threading of these processes:
2. Read the map file and display the objectives screen.
3. Read the map file and display the shop, if needed.
4. Read the map file, spawn the entities, and create the image list.
5. Load the images

Every time the map file is loaded, it forgets about what was loaded before, and it wipes out all data related to loading. This presents a problem for any background thread that’s using that data. After trying to get around this a few times, I finally got tired of trying to trick the map loader at each load point.

1. Dealing with the OpenGL variables is not thread-safe. There’s nothing in the existing code that allows two or more threads to safely talk to the GPU. There’s even a warning in the code at the point where an image is loaded:

// load the image if necessary (FIXME: not SMP safe!)

Since I know nothing about OpenGL, this was a showstopper for me. Without thread-safe code, it loops forever while trying to establish that it has or hasn’t loaded a particular image.

I created a branch in the SVN repo (branches/grayman\_loading) and committed my code changes to that, for anyone interested in looking at this. Search the code for “grayman debug” to find my changes.