UDP hole punch

There were various issues with the original code, not the least of which is that it really wouldn't have worked quite right. I changed it to not be quite so crazy but never got it done to the point of testing. One of the main problems that it has is with the timeout for sending the hole punch. It needs to happen in about 10 second intervals, at least until the a valid connection has been established. It really isn't set up to work correctly in that regard.

Asteroid related changes

I added a “bound_rad” to the asteroid field stuff. This is to allow for a size greater than 3km for determining when asteroids wrap around. It also adds 30% to this distance check to void so much “popping” by giving some extra buffer space for asteroids to reverse course rather than simple disappear/appear. This was done specifically for the WCS guys, but I never heard back as to whether it actually helped them or not (it seemed to be a nice improvement in my own tests though).

Red alert stuff

Made red alert slots dynamic. Actually changing it breaks pilot files, but this was just a precursor to a minor pilot file change to allow it in 3.6.11 or so. The reason being that there have been a lot of complaints about red alert status not being saved for certain ships, but that was only because the code will only save a certain number to the pilot file. About all that needs to happen to make this change fully active is to bump the campaign savefile version, and remove the various Assert()'s and checks again MAX_RED_ALERT_SLOTS.

Ship subsystems

Made Ship_info[].subsystems dynamic with std::vector. Pretty self-explanatory really, and a relatively small change overall.

Texture replacement

This is fundamentally changed. It all works with the notion of “sets” now. A texture “set” is kept per object, if it's actually needed, and a ptr to it is passed to the rendering code. It can be NULL if there is no special textures in use, or it will be ptr to an allocated texture_set[] struct. The reason it's done this way is so that you can maintain multiple sets of replacement textures for the same object and easily switch them out. It is also faster/cleaner to handle it this way in the model rendering code. This “set” method is also what allows for the use of the lower-level replacement textures, in ships.tbl.

What is missing for the new texture replacement setup is some that is pretty much must have: a high level manger to deal with the sets. There needs to be a system in place to give you what basically amounts to push()/pop() control of the various texture sets for a given object. Doing texture replacement should simply go through this manager, make a copy of whatever the active texture set is, make the modifications to it, then push the new set onto the stack. In order to go back to the previous set of replacement textures you would just pop the old set off the stack. The way the code is now it can deal without this, but that's only because it doesn't support replacement textures through sexps or scripting.

Ship trails

Handling ship trails is actually quite slow, considering how crap that code is. I made various changes to it to try and speed things up a bit and reduce some bad memory handling. The work involved in get those changes made and well tested, vs. the speed difference, really didn't equal out in the end. It is basically better, but it's debatable as to whether it actually makes that much difference. It's something that really needs to be put in a test build by itself to see if it actually appears to help at all or not.

Beams
Nothing major here, just added the same basic stuff from the nebula code to make beam glows fade out depending on how close to the eye they are. The primary reason for this was for beams on fighters, since the muzzle glows tend to block your view when they show up. Having them fade out still gives you the visual affect, but allows you to see through them as well.

OpenGL code
Most of this has already been committed, but with a few notable exceptions:

You'll notice two versions of gr_opengl_string(). One of them takes int's the other float's. The idea was to just go with the float version in the actual code and make an inline function in 2d.h for the int version that simply cast's to the float version, but I wasn't going to go that far with it until I was ready to commit. The reason for the change is subtle, but make a difference. The problem is with the (hackish) way we handle non-standard resolutions. Every position and size is scaled from something smaller. This makes positioning and sizing have precision issues. Some things are sized via a float scaler and others by and int scaler. This gets around most of the problems, but it can't compensate when things are actually moving on screen. Probably the best way to see the problem is on the credits screen. The normal way, at a high non-standard resolution, the text scrolling jumps and skips. Using the float version of gr_string makes it nice and smooth. Pretty basic stuff, but it makes a difference.

I set up various things to use GL_QUAD. The idea was to take advantage of cards/drivers which have an optimized/accelerated path for dealing with them. There really didn't seem to be much of a difference though, and some people even complained of negative performance changes. From a technical standpoint though, GL_QUAD and GL_QUAD_STRIP are interchangeable with GL_TRIANGLE_FAN and GL_TRIANGLE_STRIP without any data changes. So the things that I set up to use quads or quad-strips can still be done if just tell it to use tris instead.

Changes to the init code were made for AA support in Windows. It's overly complicated really, and a completely insane way to do things, but there isn't much we can do about that. Unfortunately I never could get this code tested properly on my end, my Windows box would always BSOD every other time I ran the game. I feel certain this was directly related to my failing hardware in that box, but I didn't take the chance and never activated that code for public testing in Xt builds.

GRBATCH.cpp
This code was just plain stupid. It used six tris to render anything, when it needed only 4, and it didn't work in non-HTL mode. The first change was just to make it work with 4 tris (and used GL_QUADS). I also tried to make it work in non-HTL mode, but that was never really tested very well. It also did uncool things to actually allocate/render. It uses the bitmap id as the index to look up, but there can be at most MAX_BITMAPS of those, so it's quite easy to see how a basic storage change can completely prevent allocation look-ups, and render look-ups can be optimized to an extent.

Lighting changes for warp effects
These changes are pretty minor and were intended to implement a previously discussed change whereby warp openings would use tube (directional) lights rather than simply point lights. The reason was just to give a more appropriate lighting representation to the warp effect. So instead of just lighting up an entire area it would actually shine light out of the front of the effect. In theory this was awesome, in practice, not so much. I never really got it working properly and just commented out the code as something to try and pick up again at a later time.

Ships Lab (F3 from mainhall)
The only real difference in this compared to what's in SVN is that this code deals with texture sets.

Damage lighting rendering
I made several speedups here, primarily just making it use QUAD_STRIP and having it render the entire arc at one time rather than each individual piece of the arc. Another notable change here is that I made it able to use bitmaps. You would use greyscale bitmaps, and they would get colored depending on whether it was a damage ark or and emp arc. Although rather minor changes overall, this all gave such a nice visual improvement (how it renders, not just the bitmaps) that it got some rave reviews from testers.

New gauges added to hudparse
Added talking head and ETS support for the hudparse tbl stuff. Did this for the WCS guys by request, but as usual, they don't really show much interest when things are actually done.

Model animation changes
These changes will probably be most noticeable with the docking code, since I tried to get all of that working properly by request from Vasudan Admiral. It's not 100% though, since after working on another VA request regarding modelanim, I realized that the code is absolutely horrible. I never did get what I was working on done since it was obvious that the modelanim code is just one big hackish mess and adding any solid animation support for modders requires so many weird hacks that it's just too scary. I had planned to go through and figure out how to rewrite it as part of an all new animation handling code, but never did have the time for it.

Particle code
This was mainly just a massive cleanup. It should be faster and easier to work with, but due to some performance related complaints I wasn't going to commit this without more testing. Like the grbatch changes, this is something that needs to be released in a build by itself in order to determine whether or not the reported performance issues are actually related to this code or some other changes. Another little graphical improvement here is the same alpha-fade that is used for beams, where the closer to the eye the particle is the more it fades out. This was both a visual quality change as well as a performance related change.

Joystick code
I split the API dependent joystick code between joy-win.cpp and joy-sdl.cpp (neither of which are included in the diff). This was in part a cleanup job and in part a precursor to releasing a Windows binary using SDL. I think that switching to SDL on all platforms would solve some of our strange Windows issues, no to mention cleaning up the code from a good bit of platform specific stuff. My plan was to start releasing Xt builds using SDL after 3.6.10 was released. That was on the original 6 month time-line though, so the plan obviously fell through.