Tex Mechs

An Anime-Stylized Immersion Game
Concerning Mecha and the Texas Revolution

Zach Barth

Background

The inspiration for Tex Mechs was a simplistic game I created within weeks of the release of the Nintendo Wii, "Jedi Trainer", which involved using a Wiimote connected to a PC as a "lightsaber" in a variety of virtual duels. Swinging the controller translated to gestures that were detected by the game and translated into attacks and defense positions.

When tasked to come up with a project for my Game Development 2 class, I pitched an idea for a brand new game built on this mechanic – players would become a giant, sword wielding robot, fighting through a virtual desert using a Wiimote and position-detecting floorpad as input devices. As the game thematically resembled anime shows like Gundam, we decided to adopt a matching aesthetic style; thus, the goal of this project was to establish rendering techniques to support an anime aesthetic style.

Aesthetic Style

As far as the scope of this project was concerned, the anime aesthetic that I wanted to reproduce consisted of the following stylistic elements:

- Quick bursts of action and visual movement.
- Tension from alternation between periods of action and periods of stillness.
- Solid colors for texturing, instead of patterns.
- Light cel-shaded outlines to differentiate visual objects.

In addition to rendering and animating models specifically to reinforce this style, I would have to create a number of thematic effects that could be drawn using the same style. For Tex-Mechs, these included:

- Bright and flashy general purpose explosions.
- Sword attack visualizations (as the game is primarily sword based).
- "Stream-like" lasers that appear to flow while connecting two points.
- Smoke trails to be painted behind rockets as they fly.

As I was seeking to reproduce a specific style, I needed reference art – for this I used videos and still images from the many shows in the Japanese Gundam series.

Technology Overview

As I had more important tasks ahead of me than creating a scene graph and model loaders, the OGRE (open-source graphics engine) rendering engine was chosen as the base for all graphics programming. While it uses its own formats for models, materials, animations, and particles, it supports them all out of the box and includes exporter tools for Maya, allowing us to trivially insert our artists' artwork directly into the game.

The Python language was chosen for all programming, as it has a port of OGRE available, includes many other necessary libraries (such as bluetooth and parallel port communication), and allows for a rapid and responsive development cycle. In the four hours that our game was "live" at the Game Festival, having chosen Python allowed us to rapidly fix bugs and tweak the control scheme without having to take down the entire system to recompile.

Cel-Shading Techniques

With the advent of the programmable-pipeline, cel-shading has become more common in games, to the point where it could almost be considered faux pas. The most common way of doing it involves modifying a general diffuse lighting model to clamp output into solid-color thresholds to create highlights, lowlights, and edges. Any patches with normals perpendicular to the camera are considered to be seen as “edges” and are drawn black, creating a silhouette edge around smooth objects. This technique, however, does not work well for surfaces with sharp edges, as it could cause entire faces to be clamped into silhouettes.

Alternatively, cel-shading can also be accomplished with a post-processing pass; the scene is first rendered to a texture, perhaps with depth and normal information stored alongside the color data, which is then redrawn on a fullscreen quad with a pixel shader applied. Compared to light-based cel-shading, this effect creates much sharper lines which closely resembled the Gundam source art, and can be applied to a wider variety of geometry; thus, it was the obvious choice.

Artistic Resources

While the artists creating the content for Tex Mechs were free to generally create whatever they wanted, the nature of the cel-shading techniques used required specifically designed textures. Only solid colors were allowed, as any sort of gradients or non-solid patterns would have black lines painted across them. Internal edges that were to be highlighted had to be either explicitly painted black in the texture or textured with different colors on each side of the line.

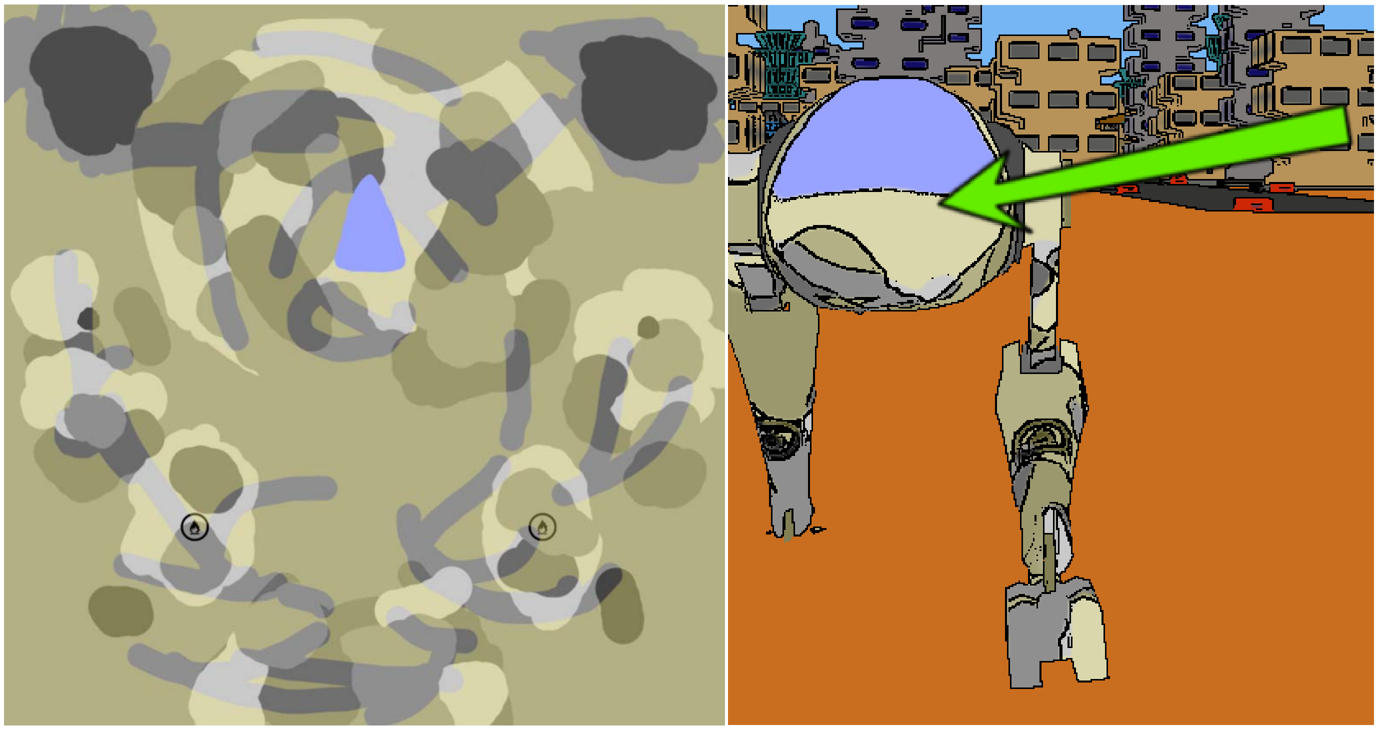


Figure 1 – Kamikaze Bot Texture and Model

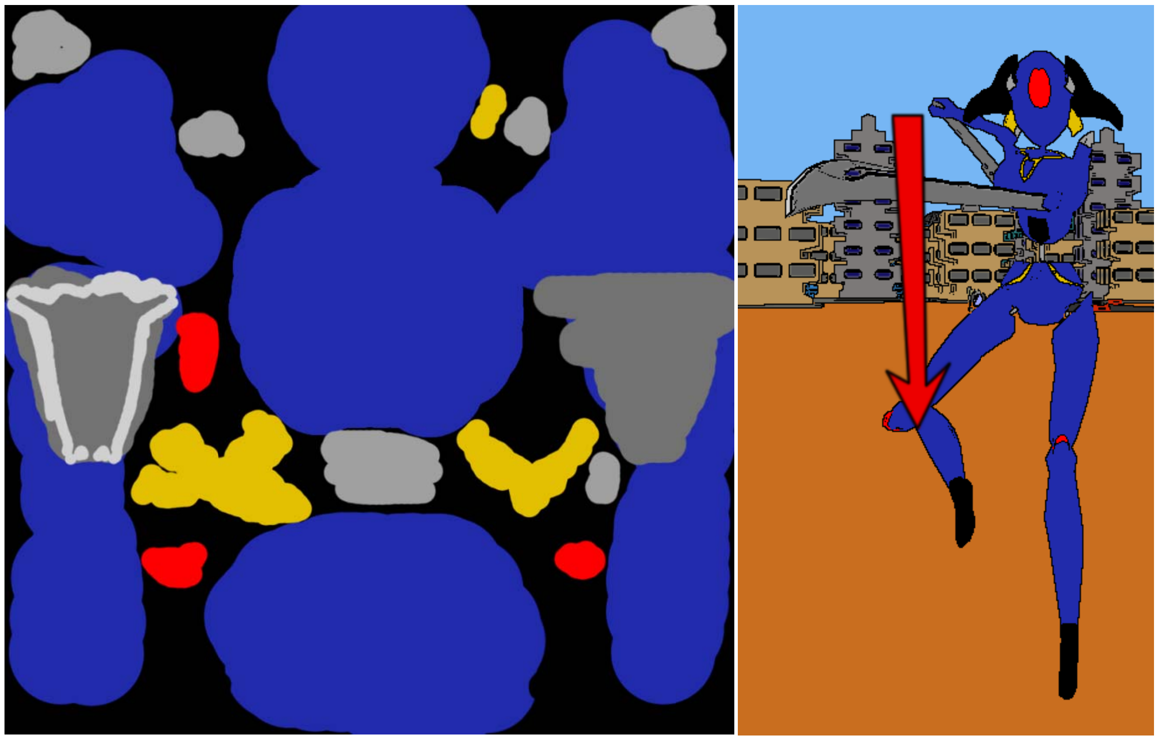


Figure 2 – Sword Bot Texture and Model

Particles

As the OGRE rendering engine comes complete with a fairly comprehensive particle system, I decided to focus on using this system to create my effects rather than trying to create one myself. Instead of using typical sorted-alpha particles, though, I decided to use color-modulated, additive textures which "add" together to create shapes of color that would then be shaded. These textures were relatively simple and grayscale, where black pixels contribute nothing and white pixels contribute the color assigned to the specific effect.

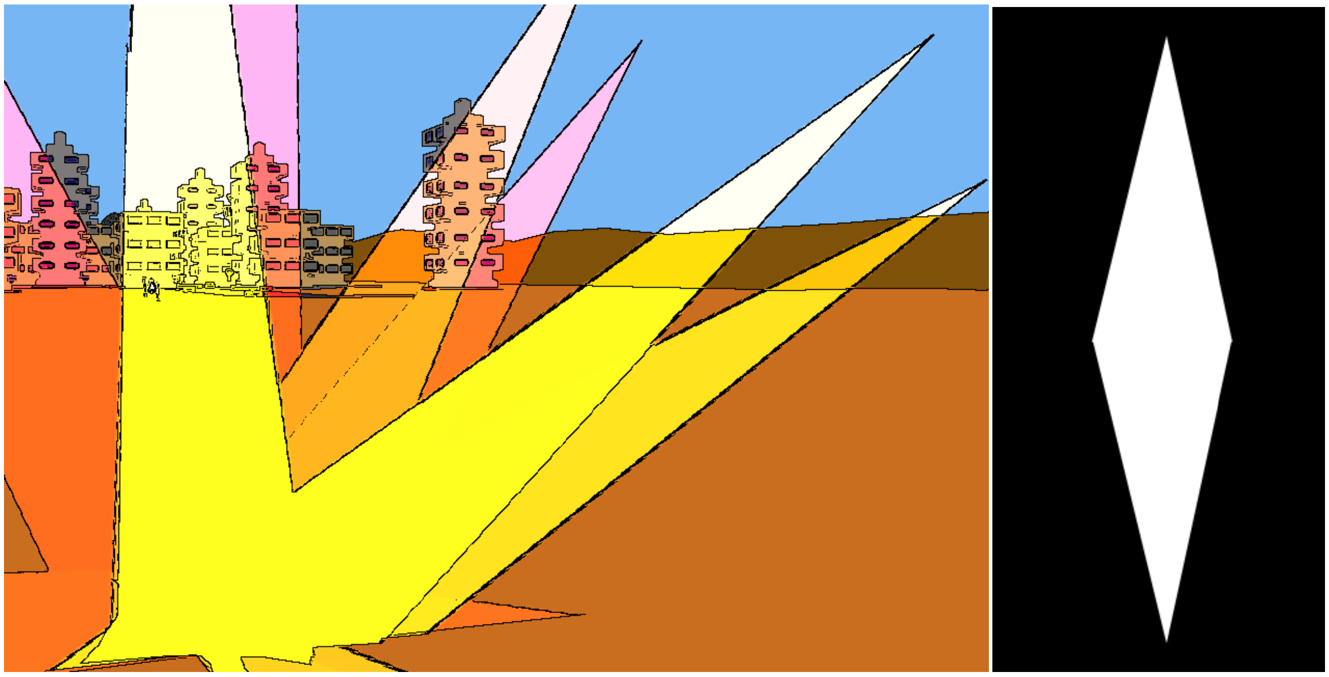


Figure 3 – Explosion Image and "Geometry Texture"

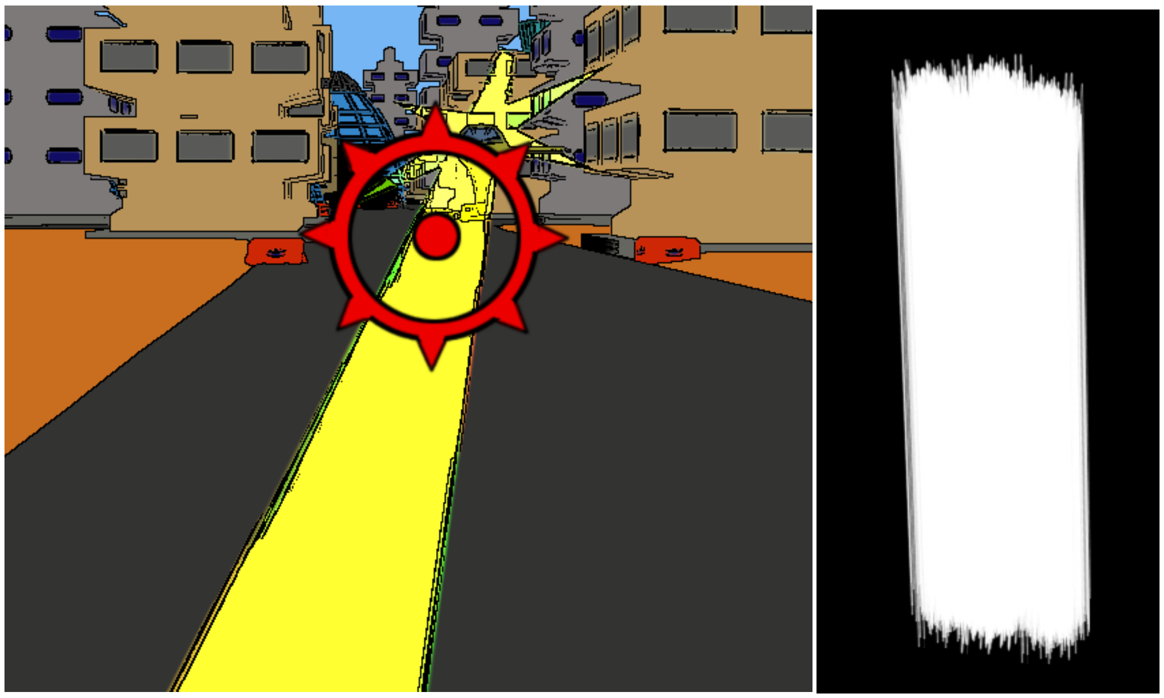


Figure 4 – Laser Image and Geometry Texture

As the OGRE particle system is quite robust, I was able to do all particle scripting from within OGRE particle script files. For explosions, diamond shaped particles were rapidly “shot” out of the emitter in all directions with very small movement speeds, and then faded away with time; the modulating color, scale, and duration were all set programmatically through the code that instanced the particle systems. The script file for creating explosions can be seen in Listing 1.

```
Effects/Explosion
{
    material Effects/Explosion
    particle_width 10
    particle_height 50
    cull_each false
    quota 10000
    billboard_type oriented_self

    emitter Point
    {
        angle 360
        emission_rate 30
        time_to_live_min 2
        time_to_live_max 4
        direction 0 0 1
        velocity 0.15
    }

    affector ColourFader
    {
        red -1
        green -1
        blue -1
    }
}
```

Listing 1 – The Explosion Particle Script

Ink Shader

The shader used to "ink" rendered scenes, as shown in Listing 2, is a modified Sobel edge-detection filter. Instead of drawing only edges, however, the shader replaces the original sampled color with black wherever an edge is present, based on a threshold hardcoded into the shader.

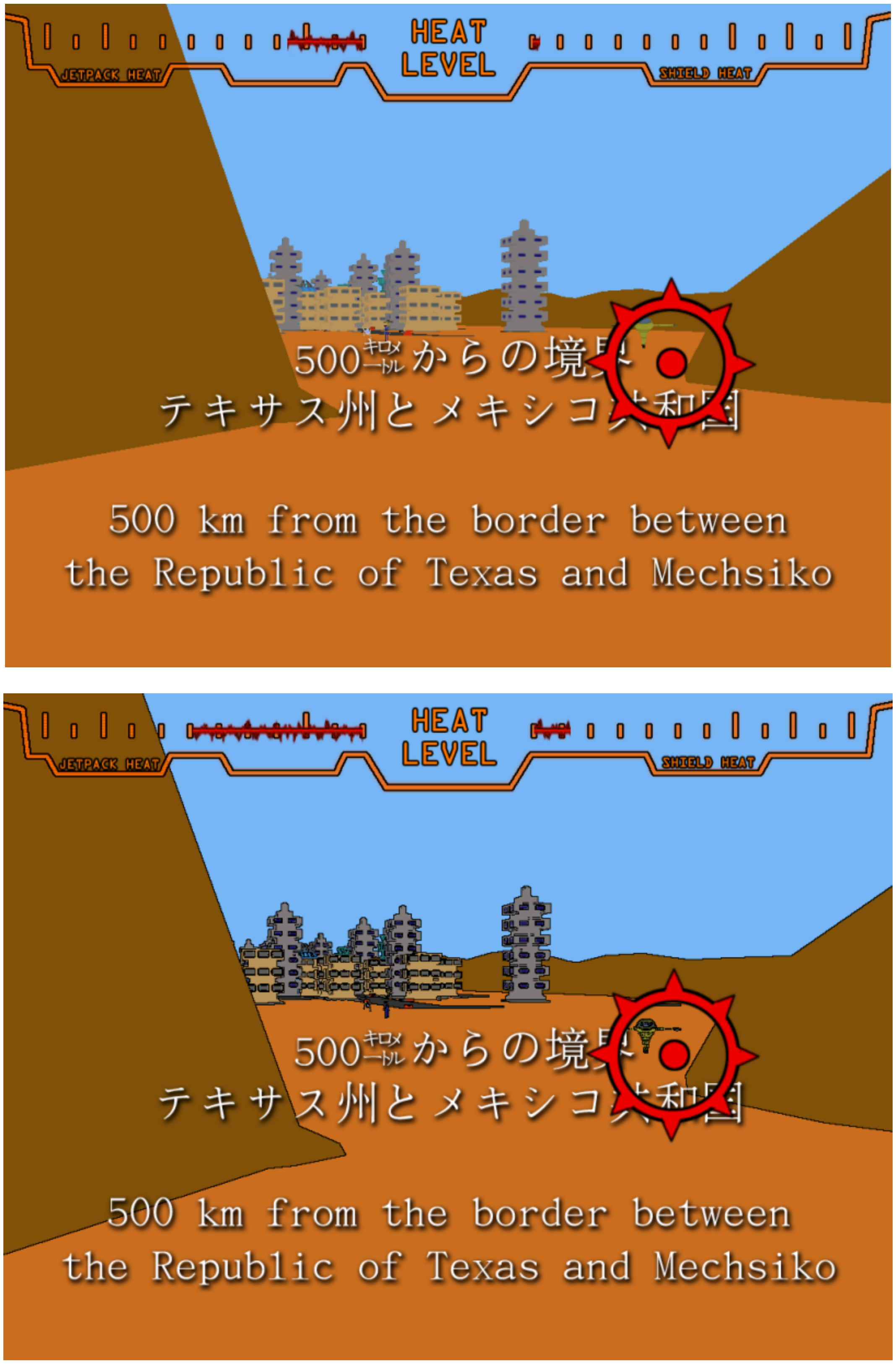

HEAT LEVEL
JETPACK HEAT
SHIELD HEAT
500キロメートルからの境界
テキサス州とメキシコ共和国
500 km from the border between
the Republic of Texas and Mechsiko
HEAT LEVEL
JETPACK HEAT
SHIELD HEAT
500キロメートルからの境界
テキサス州とメキシコ共和国
500 km from the border between
the Republic of Texas and Mechsiko

Figure 5 – Unshaded and Shaded Screenshots

```
sampler RT: register(s0);

float4 secondPass_FP(float2 iTexCoord: TEXCOORD0, uniform float2 vTexelSize) : COLOR
{
       float2 usedTexelED[8] = {
              -1, -1,
               0, -1,
               1, -1,
              -1,  0,
               1,  0,
              -1,  1,
               0,  1,
            1,  1,
       };

    float4 sample = tex2D (RT, iTexCoord);
       float4 cAvgColor= 8 * sample;

       for(int t=0; t<8; t++)
              cAvgColor -= tex2D(RT, iTexCoord + vTexelSize * usedTexelED[t]);

    float4 unit;
    unit.xyzw = 1;

    float ink = dot(cAvgColor, unit);

    if ( ink < 0.05 )
        return sample;
    else
        return 0;
}
```

Listing 2 – The Inking Shader (Cg)

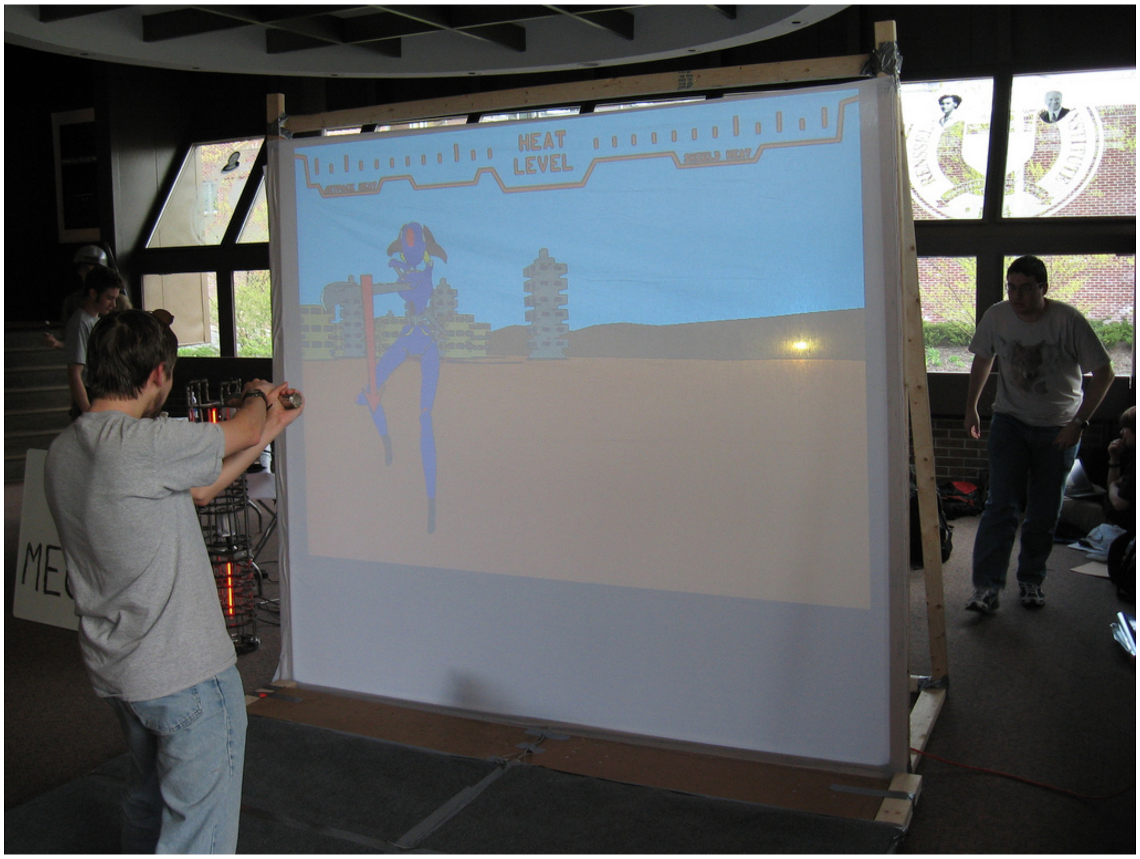
The biggest problem with detecting edges only by color is that internal overlapping, such as an arm moving in front of a torso, is only shaded if the two objects are different colors. Sampling depth and normals in addition to color would work much better for these cases, but was not viable as I couldn't figure out how to accomplish this in Ogre – as far as I know, it does not support any easy way to apply non-compositor based shaders. Luckily, due to the speed and action of the game combined with a strong use of colors, this never became a problem – our players were far too busy to notice any transiently unshaded lines.

An interesting side effect, visible only in the beginning of the game, was the shading of z-fighting artifacts. At the start of the mission, players enter high above the map and fall down onto it; this large distance, combined with the fact that our level model was made out of two overlapped ground planes, caused visible z-fighting as the player dropped to the ground. Because the ink shader affects everything rendered, large shifting black patches and jiggly black intersection lines were both clearly visible. While this could have been mitigated by redesigning the level mesh, it was never deemed important enough to warrant fixing in the last days of the project, and simply remained as a curious artifact of our special rendering techniques.

Conclusion

Although there were concerns about whether we would finish in time, my team managed to pull together in the last few days and polish Tex Mechs into an exceptional product with a comprehensive aesthetic and a strong "fun-factor", to which our unique visual style was an indispensable component.

Installation Photographs



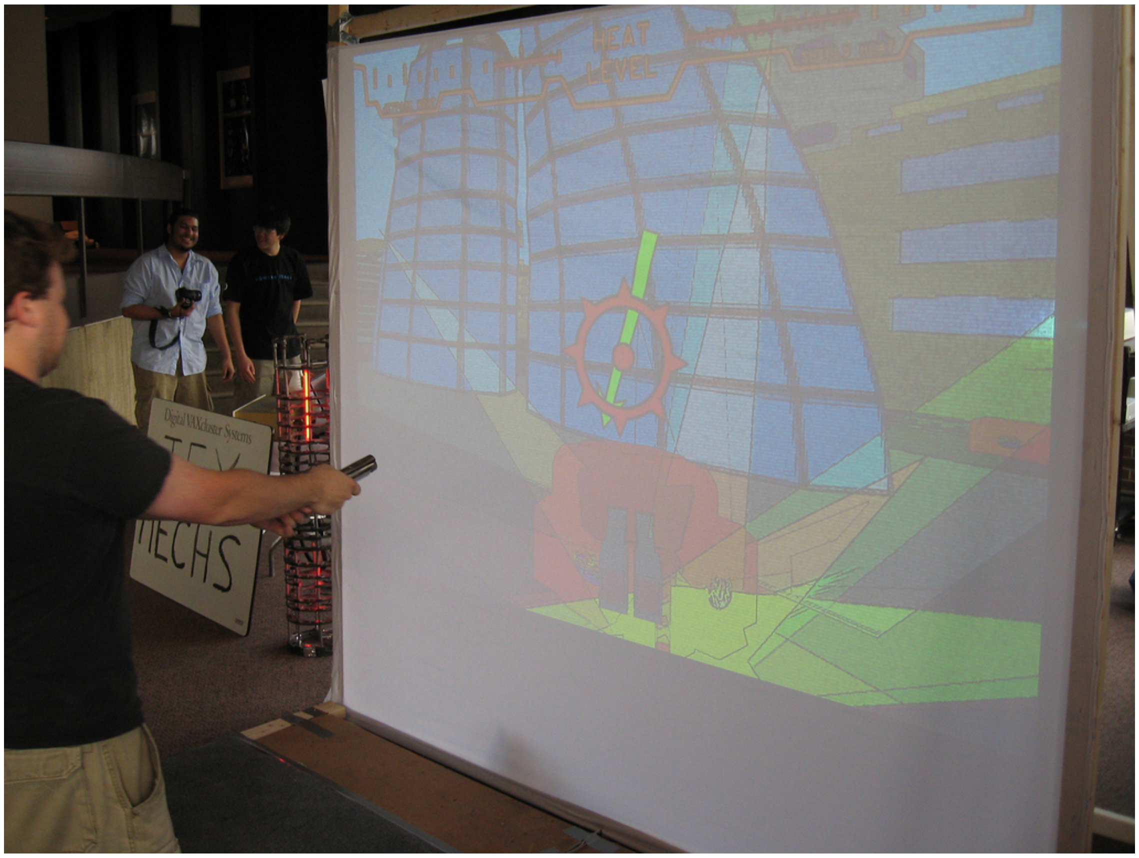


Figure 6 – People Playing Tex Mechs at the 2008 RPI Game Festival