Texture Mapping and 3-Dimensional Primitive Animation in a Pokemon Game

Amanda McNary, Erin Lavoie, and Sarah Beller
{amcnary, elavoie, sfbeller}@stanford.edu

August 14, 2015

1 Context

Our project combined several technical aspects of computer graphics with real-world applications in various fields. We wrote an expanded and improved parser to incorporate multiple filetypes and meshes, implemented three-dimensional animation, skinned a skeleton with an object mesh, and added bound-checking to our game.

Scene generation with multiple objects and textures was a primary aspect of this project. Though the scope of the previous assignments for CS148 required rendering individual objects, graphics in the real world almost always incorporate multiple objects, textures, and shaders. It goes without saying that accommodating multiple objects and files, while retaining control over each element, is critical for graphics that are at or above the intermediate level. Modern graphics generally uses programs such as Maya to render detailed objects, and our project involved some experimentation with Maya to generate a scene. Lacking the computational power that professionals enjoy, however, it became apparent that rendering complex graphics would come at the expense of speed and user experience.

An interesting characteristic of rendering multiple objects is that the objects need not be rendered in the same style, and mixing medias becomes an easy option. One common technique that is partially recreated in this assignment is the merging of photorealistic images with other images that are clearly unreal. Computer graphics (CG) are used in this way for most movies today, from Jurassic Park to Ted, and the graphics have become so advanced that is often difficult to distinguish where the computer ends and reality begins. Our assignment takes a very primitive approach to mixing media by mixing real, amateur photos, with objects that are 100% synthetic (and do not even represent real objects).

Realistic 3D animation is crucial to modern video games and movies. Over the past few decades, animation has gone from a small set of rigid limb movements to the fluid and varied motion of modern video games. As processors have sped up, animations could become more complex without sacrificing user experience. For example, over the years Mario's jump has changed from a two-pose 2D movement to a 3D one with limbs moving independently. Modern character rigs allow animators to control every part of a character while in motion. Collision detection is also important for plausible gameplay and CGI, since characters have to interact with other bodies and their surroundings in a physically realistic way. Precise and realistic animation is also necessary for scientific simulations. For example, NASA engineers rely on accurate 3D animations to project satellite trajectories prior to launch. These sytstems need to be responsive to user input while providing a helpful and correct visual aid to the engineering process. The same ability to create accurate 3D animations also allows scientists to convey their work to the public in an engaging and accessible way.

While our animation does not incorporate muscles or facial movements, we can control each bone separately while modeling a full range of body movement. We were excited about working at the intersection of user control and physics: our character skeletons have to move on command, but without breaking any physical laws or looking too unnatural. We also wanted to understand first-hand the complexity of 3D animation, since simply watching a movie or playing a video game does not necessarily convey the processes behind the character movements. Having completed our project, we have a new awareness and respect for these animations.

2 Scope

The inspiration for our product came from a desire to integrate Pokemon, a shared recreational interest from childhood, with Stanford, a shared experience from the past few years. Out initial idea was to create a Pokemon world similar to those found in the Gameboy game maps, with the twist that it would all take place on Stanford's campus. This idea was appealing because it was also easily interactive, and the concept was flexible enough that we could tailor the technical components to our individual interests.

The goals for our group evolved over the course of the project, but remained relatively static after incorporating feedback from meeting with other groups for Assignment 6. Though we initially intended to do a 3D battle scene with 2D map navigation, displaying text in a format other than through the command line seemed as if it would be a limitation, so we changed our project goals for Assignment 6. We instead decided to focus only on scene navigation, but in 3D rather than in 2D. This allowed us to focus on the technically and graphically interesting components of the project while minimizing the less relevant I/O work. The tasks that we set out for our project to achieve this goal included:

- Expansion of the Assignment 4 and 5 parser
- Rendering multiple objects with different textures and/or shaders, including texture mapping from photos
- Movement of a character around a scene with collision testing for boundedness
- 3-Dimensional character animation using matrix rotation, translation vectors, and a bone-joint structure

3 Process

Our group made a serious attempt to break the project into intermediate milestones. There were two primary tracks that we worked on concurrently for our project: character and scene rendering and navigation, and 3D animation. They will be described separately here.

3.1 Character and Scene Rendering

We were fortunate enough to find mesh files for various Pokemon online [6], but the .obj files had an expanded syntax relative to our parsers from Assignments 4 and 5, including connectivity to .mtl files, groups, and smoothing. We decided that smoothing and grouping were going to be unnecessary for the scope of our assignment, but we definitely needed to make our program compatible with multiple textures. Therefore, the first intermediate requirement was to expand our parser to be compatible with .mtl files. We wound up writing two iterations of the expanded parser because the first iteration was written for Mac, but we found out later that there was little compatibility between the installed versions of C++ and other pathnames (e.g. forward vs. backward slashes).

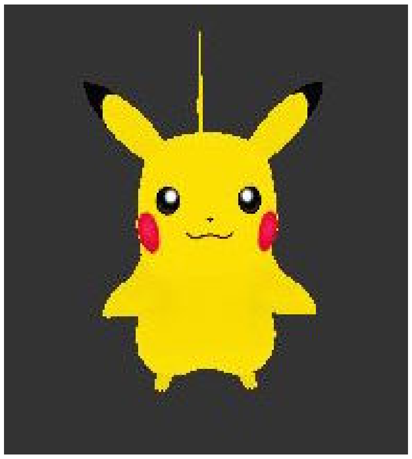

Figure 1: Pokemon rendering via .obj and .mtl files

The second initial step was to render the scene objects around the character. Before this assignment, we had only rendered one object at a time, so this was a critical, if complex, step. The first iteration of the scene was a simple box with a green floor, red walls, and blue sky. Once we had the capability to render a character inside of a scene, we created basic movement functions for both the camera view and the character itself. The result was a primitive Eevee (or another character) walking around a cartoon-like arena.

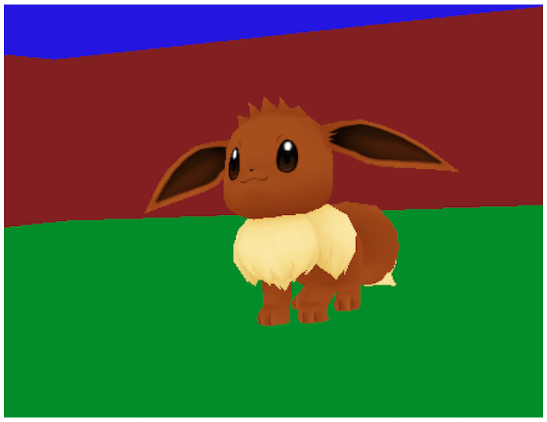

Figure 2: Eevee in the basic cartoon world

The next step required actually going to the quad and taking a few dozen test pictures. An initial thought was that it would be easiest just to take a panoramic shot, but there were many confounding factors that made this impossible, namely the tourists and the planters, which we decided not to include in our project. Instead, it was necessary to take pictures of the sides of the quad piecewise, which explains the differences in lighting and color across the four walls. Other difficulties included aligning and cropping the images to look more consistent, which was much more nuanced than we had expected beforehand. The quad was also difficult to render because it was not feasible to take a bird's-eye view of the center decal. To get around this, an image of the bricks was tiled and an image of the decal from Google maps was worked in. (Specifically, the image was found at this link using Google Maps).

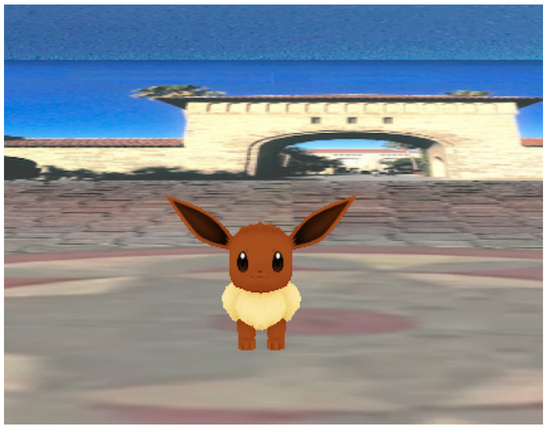

Figure 3: Initial quad with image texturing for walls

Originally, an actual image of the sky was used for the sky texture, but as the image shows, the disconnect in sky color between the different faces seriously detracted from the scene quality. Therefore, we took two distinct approaches, both of which can still be used in the final product. First, Maya was used to create 3D building representations, and after quite a bit of overhead for learning Maya, we successfully performed texture mapping on the various polygon faces. As it turned out, using the additional 3D representation of the quad walls caused such a performance decrease that it was a poor option for our final product. Instead, the sky was set to be a static color and the skies in the wall images were standardized to share the same sky color. After the standardization process, the character and scene were successfully integrated and awaited final animation.

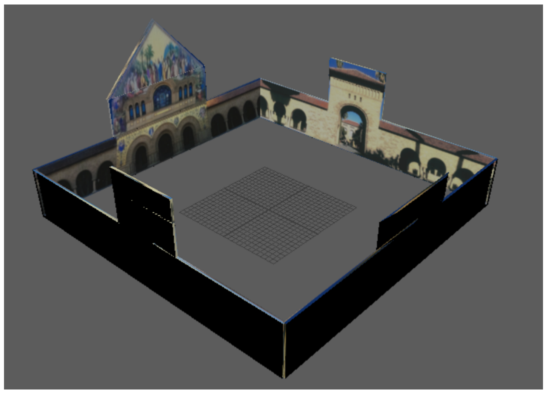
Figure 4: Screenshot of 3D representation with texture mapping in Maya

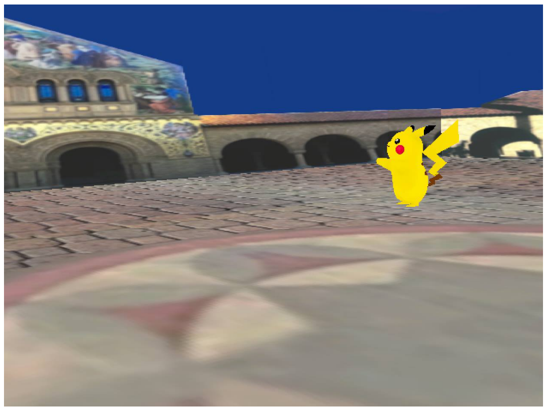
Figure 5: Final product after sky standardization

3.2 3D Animation

To make an engaging game experience, our Pokemon characters had to be able to move around in the game space. Translating and rotating the characters allowed user control, but we wanted to go a step further and animate the character meshes directly. After meeting with the TAs to discuss implementing 3D animation, it was clear this would be challenging, so we broke the animation up into several intermediate milestones to ensure there was a finished product that worked.

First, we drew a two-dimensional ball-and-stick joint model of an arm, and worked on rotating the lower arm bones around the shoulder, elbow, and wrist joints. The hierarchical nature of the bone structure led to us experimenting with several types of data structures and drawing strategies. We began with a bone struct that stored absolute position and pointed to parent bones, so that children could update their position as the parent bone moved in space. However, it was inefficient to calculate the location of each child compared to their parent continually, so we switched to the structs storing relative locations to the correct end of each bone's parent bone. With the relative orientation in space complete, we were still struggling to implement rotation around each joint, since the current struct model required us to recalculate the x and y position of each bone vertex by hand. We decided to switch to storing each bone's start position, length, and angle of rotation around each axis. With this new struct in place, bones could rotate around their parent joint with far fewer calculations. We eventually found that a bone struct with pointers to child bones allowed us to implement a recursive bone drawing method, which ensured that each child was placed relative to its parent joint and preserved the joint rotation.

Having completed the two-dimensional animation, we moved to cuboid bones in three-dimensional space. As expected, the transition from 2D to 3D was challenging. The bones could now rotate around all three axes, and still had to be responsive to movements in their parent bones. To keep the problem tied to the human body, we put in restrictive angles of movement for each bone and each axis of

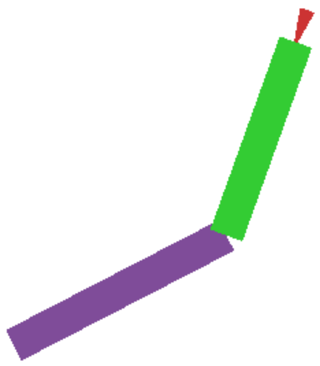

Figure 6: 2D arm with rotation around joints

rotation, to limit how far each bone could move around its parent joint.

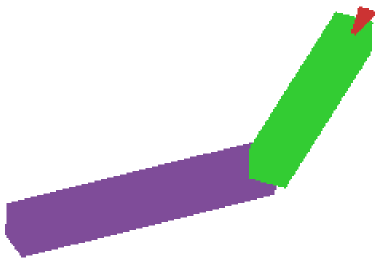

Figure 7: 3D arm with rotation around each axis

Once the three-dimensional bone drawing was complete, we drew an entire skeleton. The user could rotate each bone using keyboard commands. We then moved to animating the body automatically by creating loops of motions through time. We decided to implement keyframe interpolation, since it allowed us to specify a small number of positions and angles and then let the algorithm take care of the meat of the animation's calculations by interpolating between the known angles based on time. For the legs we made a back-and-forth motion, where the two legs move in opposite directions. Since Pikachu waddles when he walks, we added a side-to-side swaying of the torso while the legs moved forwards. We also added a side-to-side swaying of the head, which swayed in a wider arc than the torso, like a bobblehead doll.

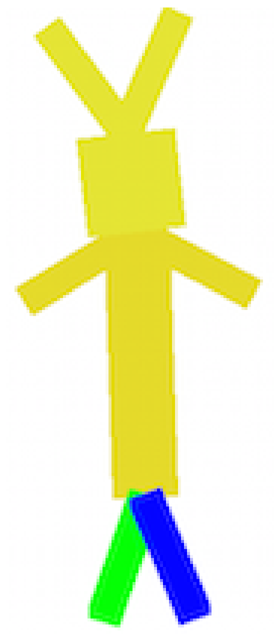

Figure 8: 3D swaying animation

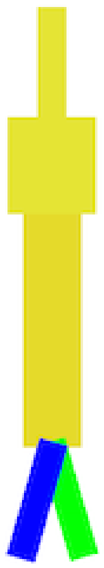

Figure 9: 3D walking animation

We then had to transfer the skeleton into the quad scene and connect it to the Pikachu object mesh. The challenge was to skin the skeleton, meaning we had to connect all the vertices in the mesh to the closest bone, so that the mesh would move correctly when the skeleton was animated. Since there were over 30,000 vertices, our first step was to connect all vertices to the spine bone.

For the sake of efficiency, we decided to store the skeleton in a vector and store the respective relevant bone index within each vertex itself. In a more abstract sense, each bone is essentially a coordinate, a rotation, and a collection of pointers to child bones. The coordinate is the point around which all vertices skinned to the bone rotate. This can be thought of as a joint. The rotation describes how the vertices should rotate about the joint. So now we can think of each mesh vertex as a vector in 3D space, with a starting coordinate at the joint and an end coordinate at the actual vertex coordinates. We want to rotate this vector by the amount specified in the bone about its start coordinate. Hence we translate the vector to the origin, multiply it with a 3x3 rotation matrix built from the information provided in the bone, and translate it back to its joint. This is done by subtracting the vertex coordinates piecewise by the joint coordinates, effectively translating it to the origin, applying matrix multiplication, and then adding the new values obtained piecewise with the joint coordinates.

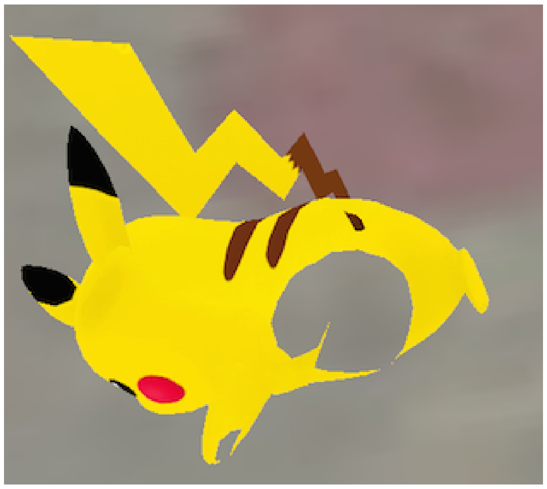
Figure 10: Pikachu sinking below ground level when first attached to the spine.

4 Achievement

Our project objectives initially presented in our proposal changed significantly after meeting with the TA and other groups for feedback. Upon addressing issues that were brought up, we recalibrated our final goal to be within a feasible scope. Following this shift we outlined several tasks that we would need to complete as building blocks for the final product. These include texture mapping multiple textures to a single surface, reading a .mtl file, further developing our keyboard commands, collision detection, and animation. We were successful in making progress in each of these areas, although the extent varies and at times stands independently from visible progress. Our biggest struggle was integrating the code for animating the simple skeleton with the Pikachu wireframe mesh. When starting this aspect of the project, we were well aware that 3D animation was a non-trivial challenge that could ensnare us in an impossible bug or glitch and prevent us from having any tangible work to present. We took preventative measures for this possibility by setting up small incremental steps towards the final goal. After relative success with creating the simple skeleton and getting all the simple animations we wished to use working, we hit our predicted roadblock. Our first simple goal in integration was to get a single bone to cooperate within Pikachu. This was significantly more time consuming than we had hoped, so we worked to reach a simplified stopping point in the overarching animation goal, where Pikachu sways while walking, but without the independent limb movements we had achieved in the skeleton. Consequently, we still have a functional version of our desired goal that is presentable and a good representation of our work in this area.

References

[1] ROEStudios, 3D Pokemon Models. Property of Nintendo/GameFreak.

[2] gamedev.stackexchange.com, Using multiple shaders.

[3] MegaBcoyle, Maya 3D Modelling Tutorial. YouTube.

[4] OpenGL Wiki, OpenGL Skeletal Animation.

[5] OpenGL Tutorial, Basic Bones System.

[6] Ohio State University, Guidance for Obj Parser.

Note: reference titles are hyperlinked.