Zirui Huang

Implemented Techniques

The following interpolation techniques were successfully implemented:

- Bézier Euler Interpolation
- SLERP Quaternion Interpolation
- Bézier SLERP Quaternion Interpolation

Observation

Bézier Euler Interpolation

For n = 20, using a dance motion as an example, Bézier Euler interpolation performs similarly to linear interpolation in terms of general motion trends. For small-scale dance movements, this method works well, providing smooth bone rotations and translations without noticeable unnatural behavior. However, certain scenarios still exhibit unnatural rotational artifacts:

- During small turns (e.g., rotating from a side view to a front-facing view), the skeleton model experiences unnatural overall rotation, leading to an unintended side-to-side sway.
- In large motions such as a 360-degree jump spin, slight unnatural jitter appears at takeoff, while the landing phase exhibits significant oscillation and instability.

From Graph 1, I observe that while the overall trends of Bézier and linear interpolation align, Bézier produces a visibly smoother curve. Compared to the input motion, the general trend remains similar. However, between frames 630 to 670, where the input motion follows a downward then upward trajectory, Bézier Euler simplifies this into a continuous downward motion, introducing a significant deviation.

For n = 40, Bézier Euler interpolation performs significantly worse. Using a martial arts motion sequence as an example, the motion contains numerous instances of extreme unnatural rotations. These issues are especially prominent when the character undergoes side-to-side turns or large-angle rotations, causing exaggerated incorrect tilting and swaying, almost making the character appear to be flipping over. Additionally, at certain moments, wrist rotations appear highly distorted and unnatural

SLERP Quaternion Interpolation

Compared to Bézier Euler interpolation, SLERP quaternion interpolation provides significantly better results.

For n = 20, using the same dance motion as before, SLERP eliminates the unnatural jitter and rotational artifacts observed in Bézier Euler interpolation. The transitions between movements are smooth, and there are no abrupt or awkward rotations.

However, one downside of SLERP interpolation is that the overall motion appears somewhat rigid. The transitions between movements feel stiff. Additionally, one noticeable unnatural effect occurs during the landing phase after a jump—a brief discontinuity in displacement of the character makes the movement appear slightly unnatural.

For n = 40, using the martial arts motion, SLERP performs exceptionally well in maintaining stability, avoiding the exaggerated unnatural rotations seen in Euler-based interpolation. However, the primary drawback remains: the motion feels highly linear and lacks organic smoothness, making the animation appear somewhat mechanical.

From Graph 2, SLERP shows similar curve alignment with input motion as linear Euler interpolation. When rotation magnitudes are small, SLERP aligns well with input motion. However, in larger rotational changes, the deviation between SLERP and input motion follows a pattern similar to linear Euler.

From Graph 3, SLERP clearly outperforms linear Euler interpolation. When the overall angle variations are small, both SLERP and linear Euler interpolation align closely with the input motion curve, but SLERP demonstrates a higher degree of accuracy. Notably, in the final few frames of this motion sequence, SLERP maintains a smooth trajectory that closely follows the input motion, whereas linear Euler interpolation exhibits a highly linear downward trend, deviating more significantly from the expected motion.

Bézier SLERP Quaternion Interpolation

For n = 20, using the same dance motion, Bézier SLERP quaternion interpolation shows a clear improvement over SLERP. The animation appears much smoother, with highly natural angular transitions. The overall movement feels more expressive, significantly reducing the stiffness observed in SLERP. This method maintains interpolation stability, avoiding unnatural displacements or sudden rotations, while also enhancing the overall fluidity.

From Graph 4, it is evident that Bézier SLERP quaternion interpolation aligns well with the input motion, with only a noticeable deviation between frames 225 and 300. However, even in this range, the accuracy is better than Bézier Euler interpolation. Particularly in the final ten frames, Bézier SLERP quaternion interpolation achieves a very high level of fidelity to the input motion, whereas Bézier Euler exhibits significant deviations.

For n = 40, in the martial arts motion, Bézier SLERP quaternion interpolation produces the smoothest transitions among all tested methods while remaining stable. However, a notable downside in this scenario is unnatural positional shifts, where small sliding or jittering effects make the motion appear as though the character is gliding or drifting across the ground.

Summary of Strengths and Weaknesses

Bézier Euler Interpolation

Strengths:

- Provides smoother interpolation compared to linear Euler.
- Works well for small-scale movements like subtle dance animations.
- Produces visibly smoother curves compared to linear interpolation.
- Computationally inexpensive and fastest among the three methods,

Weaknesses:

- Unnatural rotational artifacts appear, especially in side-to-side turns and large-angle rotations.
- Jitter and oscillation in fast movements, such as a 360-degree jump spin.
- For n = 40, introduces severe unnatural rotations, excessive tilting, swaying, and wrist distortions.
- Gimbal lock risk due to direct Euler angle interpolation.

SLERP Quaternion Interpolation

Strengths:

- Eliminates the unnatural jitter and rotational artifacts seen in Bézier Euler.
- More stable and natural transitions, avoiding abrupt or awkward rotations.

Weaknesses:

- The motion appears stiff and rigid.
- Discontinuity in displacement during certain motions, such as jump landings.
- In higher frame interval settings, the animation feels too linear and lacks organic variation.
- More expensive than Euler interpolation, as it requires trigonometric functions for spherical interpolation.

Bézier SLERP Quaternion Interpolation

Strengths:

- Provides the smoothest and most natural interpolation among all implemented methods.
- Eliminates stiffness from SLERP while maintaining stability.
- Produces fluid and expressive motion
- Outperforms Bézier Euler in accuracy, especially in large rotations.

Weaknesses:

- Small positional artifacts, leading to sliding or drifting effects on the ground.
- More computationally expensive due to multiple SLERP computations per frame.

Extra Credits

Initially, I planned to import a 3D model for skinning but eventually abandoned the attempt. I installed the Assimp library and GLEW, converted the Assimp to Win32, and wrote model-loading code (displayModel.cpp), but ultimately, it was not used.

I also attempted to introduce custom shaders, configuring GLM for matrix calculations and STB for texture loading. I implemented a shader loader (shaderLoader.cpp) and wrote basic GLSL code, but due to time issues, I abandoned the effort midway.
The only successful addition was sampling a texture using STB and rendering the floor with openGL fixed pipeline, along with adjusting some lighting parameters for improved visual quality.
I implemented a LoadFloorTexture in mocapPlayer.cpp,

```
void LoadFloorTexture(const char* filePath)
{
    int width, height, nrChannels; // # of channels
    // load floor texture
    unsigned char* data = stbi_load(filePath, &width, &height, &nrChannels, 0);
    if (!data) {
        std::cerr << "Failed to load floor texture: " << filePath << std::endl;
        return;
    }

    glGenTextures(1, &floorTexture);
    glBindTexture(GL_TEXTURE_2D, floorTexture);

    // set texture parameters
    // set repeat when exceeds range
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT); // horizontal (u)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT); // verticle (v)
    // scale method
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    // send data to GPU
    GLenum format = (nrChannels == 4) ? GL_RGBA : GL_RGB;
    glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);

    stbi_image_free(data);
    glBindTexture(GL_TEXTURE_2D, 0); // unbind

}
```

and modified RenderGroundPlane to render the floor with texture

```
for(int i=0; i<planeResolution; i++)
  for(int j=0; j<planeResolution; j++)
  {
    float planeAmbientAct[4] = { (float)(ambientFskeleton * rPlane), (float)(ambientFskeleton * gPlane), (float)(ambientFskeleton * bPlane), 1.0f};
    /*float factor = (((i+j) % 2) == 0) ? 0.5f : 1.0f;
    planeAmbientAct[0] *= factor;
    planeAmbientAct[1] *= factor;
    planeAmbientAct[2] *= factor;
    planeAmbientAct[3] *= factor;*/
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, planeAmbientAct);
    glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f); glVertex3f((float)(-groundPlaneSize/2 + i * planeIncrement), (float)groundPlaneHeight, (float)(-groundPlaneSize/2 + j * planeIncrement));
    glTexCoord2f(1.0f, 0.0f); glVertex3f((float)(-groundPlaneSize/2 + i * planeIncrement), (float)groundPlaneHeight, (float)(-groundPlaneSize/2 + (j+1) * planeIncrement));
    glTexCoord2f(1.0f, 1.0f); glVertex3f((float)(-groundPlaneSize/2 + (i+1) * planeIncrement), (float)groundPlaneHeight, (float)(-groundPlaneSize/2 + (j+1) * planeIncrement));
    glTexCoord2f(0.0f, 1.0f); glVertex3f((float)(-groundPlaneSize/2 + (i+1) * planeIncrement), (float)groundPlaneHeight, (float)(-groundPlaneSize/2 + j * planeIncrement));
    glEnd();
  }

  glBindTexture(GL_TEXTURE_2D, 0); // unbind texture
  glDisable(GL_TEXTURE_2D);

  glDisable(GL_POLYGON_OFFSET_FILL);
```