CS 284A: Final project- Leap and Jump Milestone

Yifeng Zhang
yifengzhang@berkeley.edu

Sihua Ren
771191795rsh@berkeley.edu

Aslina Cheng
aslina_cheng@berkeley.edu

Boyuan Ma
boyuan.ma@berkeley.edu

ABSTRACT

In this project, we aim to enhance an existing platform jumping game by incorporating texture and light rendering techniques. The interactive application will allow users to control the height of their jumps by adjusting the duration of their screen taps, enabling them to achieve higher scores. This enhancement will be complemented by captivating animations and engaging background music.

KEYWORDS

Graphics, Texture, Rendering, Interaction, Game

ACM Reference Format:
Yifeng Zhang, Sihua Ren, Aslina Cheng, and Boyuan Ma. 2018. CS 284A: Final project- Leap and Jump Milestone. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXX X.XXXXXXX

1 INTRODUCTION

Originally, the game "tiao-yi-tiao" was a basic platform jumper featuring a monochrome character model and geometric block designs, which many found plain. Our goal is to revitalize the game by enhancing its visual appeal through texture and light rendering.

We will construct a virtual scene using hierarchical modeling techniques, comprised of multiple virtual objects. And We plan to apply texture mapping to at least two main objects within the scene, enhancing their visual detail and realism. For the evaluation, we will measure the frames per second (FPS) during interaction and scene manipulation to ensure smooth real-time performance.

When the project finished, users will be able to control and transform objects within the scene, such as moving, rotating, and scaling, using keyboard or mouse inputs.

2 PROCESS

2.1 Basic concept

2.1.1 OpenGL:

OpenGL (Open Graphics Library) is a cross-platform, cross-language application programming interface (API) used for rendering 2D and 3D vector graphics. It's widely used in computer graphics for applications such as video games, simulations, and CAD (computer-aided design).

2.1.2 Lights and shadows:

With a scene and a camera set up, we can display 3D models, but the models may appear distorted. This distortion often happens because the color at each position of the model is uniform. In reality, the actual appearance of a model's color is complex, influenced by factors such as the color and angle of lighting, the distance to the light source, and the object's own color and material. While simulating real-world lighting characteristics can be complex, it's not necessary for them to be perfectly accurate, and simplifications can be made.

A common simplification used in 3D modeling is the combination of ambient light and directional light.

Ambient light simulates diffuse reflection of light. It has no direction and uniformly illuminates all objects in the scene.

Directional light simulates sunlight. Assuming the sun is infinitely far away, the light rays from it are parallel. The addition of directional light provides a sense of depth as surfaces facing the light become brighter and those away from it become darker.

Besides directional light, other light sources like point lights, area lights, and spotlights can also be used. Each affects objects differently; for example, point lights emit light in all directions, and spotlights illuminate only objects within a specific range. However, the underlying principle for all these light types is the same: they utilize the reflection of light to create variations in brightness and darkness on objects.

2.1.3 texture mapping:

Texture mapping is a technique used in computer graphics to add detail, surface texture, or color to a 3D model. It involves applying a 2D image, known as a texture, onto the surface of a 3D shape. This process enhances the realism of the 3D object without increasing the complexity of the geometry.

2.2 Set up the scene

2.2.1 Coordinate system:

To construct a scene, we must first determine the world coordinate system. The most intuitive thing to do is to use the three.js coordinate system. As shown below, red represents the X-axis, green represents the Y-axis, and blue represents the Z-axis. The objects we create with three.js are located at the origin of this coordinate system.

2.2.2 Camera:

Looking at the "jump", we find that the object can only jump in two fixed directions, in order to facilitate calculation, we define these two directions as the positive direction of the X axis and the negative direction of the Z axis. The bottom is defined as the XZ plane. In this way, the position of the camera can be basically

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

determined, that is, in the negative direction of the X axis, and in the positive direction of the Y and Z. The size of the cube in "jump" has nothing to do with distance, and should use an orthogonal camera with the camera facing the origin of the coordinates.

2.2.3 Boxes:

There are several types of boxes in "Jump", but let's leave it for now and simplify it to a cube with a fixed height and the same length and width.

2.2.4 Plane:

If we want to show shadows, we must have a ground that can carry shadows, so we need to draw a plane large enough in the XZ plane that does not need to have color, the only purpose is to show shadows.

three.js encapsulates a class called PlaneGeometry for generating plane geometry. We can generate a square for the ground. In terms of materials, ShadowMaterial fits the bill perfectly, it can receive shadows but is otherwise completely transparent.

2.2.5 Shadows:

By default, three.js does not generate shadows. We need to do some configuration.

First, we need to alter the lighting to add shaded areas for parallel light.

After setting up the light, you also need to set up the object that can create the shadow, that is, the box. To make an object cast a shadow is relatively simple, we just need to configure the castShadow property.

2.2.6 Body:

We cannot simply splice several shapes into a whole. The reason is that in three.js, the basic operations that objects can perform are: displacement, rotation, and scaling. All of these operations depend on the coordinate system of the current object. For example, if we want to compress the current, that is, reverse compress at Y, the object will shrink toward the object axis XZ plane, and eventually will shrink to the XZ plane. If we splice multiple shapes into a whole, then the operation on the object will act on the whole object. Analyze the behavior of small people who "jump", when it accumulates force, the body is compressed, and the head is not compressed, so it is obvious that the head and torso should be able to control separately.

2.3 Charge and Jump

2.3.1 Animation:

Once we can render a static scene, it's much easier to make it move. In fact, it is the principle of animation, rendering a different image at intervals. To be more professional, we call it a "render loop" or an "animate loop."

Here we use requestAnimationFrame to create a loop that allows the renderer to draw the scene with each screen refresh, the number of screen refreshes per second, called the frame rate (typically screen frame rate is 60). There are two advantages to using requestAnimationFrame. The first is that requestAnimationFrame renders according to the frame rate. Different machines have different frame rates. For poor performance machines, the number of times to render is less, which can prevent the page from getting stuck. Second, requestAnimationFrame pauses when the screen is minimized or when it switches to another TAB, so it doesn't waste the user's valuable processor resources.

2.3.2 Charge:

The accumulation behavior of "jump" is actually the scaling behavior of the box and the torso, the box is contracted longitudewise, and the torso of the little man is contracted longitudewise and expanded horizontally. It should also be noted that because the box is shorter, the little man needs to move down as a whole, and because the torso is shorter, the head of the little man also needs to move down. The scaling behavior has a maximum value beyond which no more deformation occurs.

We set the box scaling ratio to 0.6, the torso scaling ratio to 0.6, and the torso lateral expansion ratio to 1.3. The current position of the body is actually the current height of the box, and the position of the head is the height of the body scaling minus the initial position of the head.

We use linear animation and set the maximum time. It should be noted that we should always judge the state of the villain in the animation, that is, judge whether the current click (touch) is over, once it is over, the accumulator animation will stop immediately.

2.3.3 Jump:

Our idea is that the body always jumps towards the center point of the next box, and we can calculate the Angle of the jump based on the current position of the body and the position of the next box. The jump distance of the figure is proportional to the compression of the body, which is recorded when compressing the animation. So you have the Angle and the initial velocity, and you can easily get the position of the body after the jump.

2.4 Particles and Trails

2.4.1 Particles:

The particles in "Jump" are actually two-dimensional circles, not spheres, which can be seen if you look carefully. Circles are much easier to render than spheres, and three.js also has CircleGeometry built in for rendering circular buffer geometry. In addition to their shape, their color is also characteristic, not a solid color, but gradually transparent from the middle to the sides. We can find a round image with a center that is gradually transparent to all sides and load it into the project as a particle map. As we accumulate power, if we keep creating new particles, we can expect some performance problems. The idea of optimization is also relatively simple, that is, the particles are pre-generated, placed in a pool, and reused. When we initialize, we generate a certain number of white and green particles and maintain them in an array. It should be noted that the generated particle faces the XY plane by default, which looks like an oval from our current perspective, so we rotate the particle so that it faces the camera, so that it looks like a circle.

2.4.2 Trails:

Tail effect is very common in the game, such as bullet tail, aircraft tail, etc., it has a common feature is: the shape is related to the trajectory of movement. Therefore, there is no fixed shape of the tail, and there is no ready-made map available.

The production of the tail has a fixed pattern, that is, the use of small pieces of rectangle to simulate, so that whether it is to do complex tracks, or color gradients, or even shape gradients, it can be considered to be the adjustment of a series of rectangles. In "Jump",

the trailing track is the track of the jumping figure, more accurately the track at the bottom of the figure, the shape is narrower and narrower.

The way we think about it is this. We form some very narrow rectangles as the basic components of the tail, which we call the tail pieces. When executing the jump animation, record the position of the current frame and the previous frame, place the trawled fragment in the current position, and Orient it towards the previous frame. In each frame, it is necessary to update the existing trailing debris, the update is based on the existence time, the longer the time, the smaller the height of the rectangle, until the fragment height is 0, at which time the fragment is recovered and continued to use next time.

3 FUTURE WORK

3.1 Behavior after jump

3.1.1 Landing point judgment:

We first analyze several situations after jumping, according to the distance is divided into the following categories: (1) The jump was short, and the body was still on the jump box. (2) It jumps between the current box and the second box.(3) Jump to the next box.(4) Jumped out of the next box.

Obviously, (2) and (4) are the two things that will end the game. There is a more troublesome problem here, that is, to consider the animation of the body falling to the ground, if the body jumps to the edge of the box, it should slide from the box, the body will fall. And if you jump straight to the ground, you won't tip over.

So, we're going to add three new states to the previous classification: (1) Jump to the edge of the current box and drop forward.(2) Jump to the edge of the next box (near the current box) and fall backwards.(3) Jump to the edge of the next box (away from the current box) and fall forward.

3.1.2 Buffer drop:

If it falls between two boxes, or beyond the second box, and does not touch either, then the body will slowly fall to the ground.

3.1.3 Edge drop:

Edge drop is another composite animation. It's a combination of rotation, descent and translation. When the body lands on the edge of the box, the body will start to lean, while moving away from the box as a whole, and when it is completely free of the box, the body will start to fall.

We assume that the body and the box will not collide, slide and fall are considered as an animation, the body tilt 90 degrees is an animation, set the same time of the two, then the moment the body falls, the body just "lying flat" on the ground, which is a relatively simple way to fall.

Whether it is forward inclined or backward inclined, the Angle of rotation is not the same, and the overall process is consistent.

3.1.4 Collision detection:

Almost all graphics libraries have built in the same collision detection method, which is to start from the center of the object, emit a ray to each vertex, and then check whether the ray intersects with other objects. If an intersection occurs, check the distance between the nearest intersection and the origin of the ray. If this distance is less than the distance between the origin of the ray and the vertex of the object, then a collision has occurred.

3.2 The texture of Boxes

We plan to customize some box texture to make the game more cool and playable. We are still consulting the information on how to map the box

3.3 Performance monitoring

We plan to use Performance API to monitor the performance of our project, which can help us to get highly accurate website performance data.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009