# CS 284A: Final project- Leap and Jump Final Report

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 modeled it on a little game called "tiao yi tiao". Compared with the original game, we first enhanced the color performance to make it more impactful and attractive to users, and then added a variety of texture and bump elements to make it more interesting and more expressive; One thing worth mentioning is that we replaced the particle with oski's head, which is very interesting.

## KEYWORDS

Graphic, three.js, texture, bump

**ACM Reference Format:**
Yifeng Zhang, Sihua Ren, Aslina Cheng, and Boyuan Ma. 2018. CS 284A: Final project- Leap and Jump Final Report. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

With the enrichment of people's entertainment life, more and more small programs and small games pour into the market. And "tiao yi tiao" is one of them. This is a small game that controls how hard the littleman jump to get better scores by controlling how long the users click on the screen. However, its original version is a little boring when you try many times. Therefore, we decide to make it more interesting and attracting. First, we built a similar scene to the original version. Then we update many parts like particles, trails, shading, bump mapping, light and texture. To achieve our goal, we mainly use ThreeJS and tools mentioned in the lecture. After our modification, it becomes very attractive and lively.

## 2 TECHNICAL APPROACH

### 2.1 Basic Concept:

In this project, the main tool library we use is ThreeJS. It is a popular and powerful JavaScript library that is used to create and display animated 3D graphics in a web browser. It provides a set of tools that simplify the process of rendering 3D scenes directly in the browser without the need for specialized software or heavy plugins, using the WebGL rendering API.

### 2.2 Scene Set Up:

In this part, we use the coordinate included in the ThreeJS and define the width of scene is 100, making other object based on it. Then we define a config document to store and adjust most constants used in whole code.

Looking the previous "tiao yi tiao", we find that the littleman 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 determined, that is, in the negative direction of the X axis, and in the positive direction of the Y and Z.

For the box, we first simplify it to a cube with a fixed height and the same length and width. After the maximum and minimum values of the box are determined, one of the values is randomly chosen as the length and width of the box.

For the Littleman, we find in the "tiao yi tiao", the head of it is a sphere, and the torso is composed of multiple models, from top to bottom as a sphere, with a narrow cylinder at the top and a narrow cylinder at the bottom. But in this project, we can not simply splice several shapes into a whole. The reason is that in ThreeJS, 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. We also find that when the littleman accumulates power, the body is compressed, but the head is not compressed, so obviously, the head and the trunk need to be controlled separately. So we need to create a mesh for the head alone and take the body as a whole, moving the generated geometry to the appropriate position and create the torso in three. Since the body is directly "new", its coordinate origin is the world coordinate origin. So we're going to have a littleman that's above that point, because when we created the geometry, we shifted the geometry up. If we compress the body as a whole, then the object will shrink towards the origin. If we rotate the body, the littleman will also rotate around the origin. This is obviously wrong. We need a littleman who can flip along the center of gravity. Instead of changing the axis of the body, we add one more axis system to it. That is, the whole axis is moved up to the center point, and then the added object is moved down by the same distance, so that the object is equivalent to not moving, but the origin of the axis does become the center point. So far, we can create a complete littleman.(When


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

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

create the littleman, we tried to make a wing for it, but failed. Due to the time, we may try it again later.)

## 2.3 Particles and trails

In the previous "tiao yi tiao", we can find its particles are some spheres. When we create the particles, we first want to use the "CircleGermetry" in ThreeJS to render, but we failed. Therefore, we use some interesting figures as the texture of particles. Then we can use "PlaneGeometry" to render rectangle.

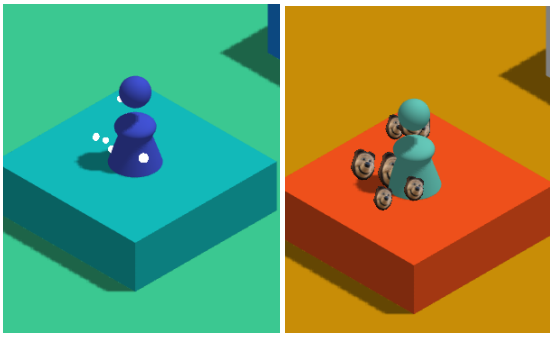

**Figure 1: Before texture and after for particles**

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 particles and maintain them in an array.
We also add the dispersion of particles. The dispersion occurs after the jump is over, and scattered particles appear at the bottom of the littleman. The dispersion of particles is only once, there is no need to loop.

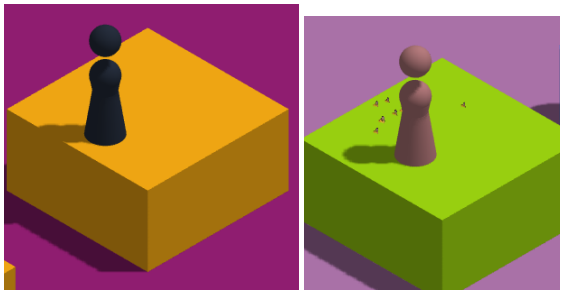

**Figure 2: Before dispersion and after dispersion**

For the tail, in the previous "tiao yi tiao", it is white and unobvious. After our modification, it becomes colorful and more obvious.

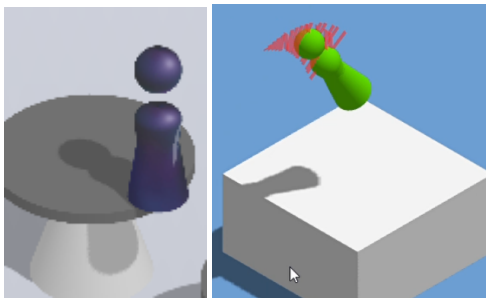

**Figure 3: Before tail and after tail**

## 2.4 Shading and materials

The objects involved in this game include the jumper character, various square and cylinder boxes, and texture mapped boxes. To make the appearance of these objects more delicate, we investigated different shading approaches. The ThreeJS library provides nice APIs that wrap around some core GLSL shaders. It can render lambertian surfaces and blinn-phong reflections with fast GLSL implementations.

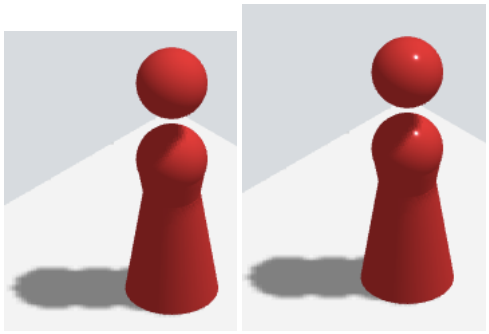

**Figure 4: Lambertian and Blinn-Phong shading of the game character**

For the texture mapped objects, we added bump mapping and displacement mapping to further demonstrate the realistic lighting effects on the objects. For example, the Rubik's cube texture that we used can be greatly enhanced by bump and displacement mapping which demonstrate the realistic lighting effect on a 3D Rubik's cube with the color tiles embedded a bit inward of the white frame.

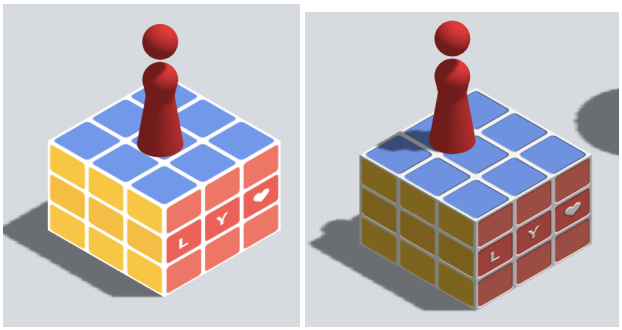

**Figure 5: The effect of bump and displacement mapping**

Another example of it applied to a delivery box texture shows the realism of the seal and tags on the box.

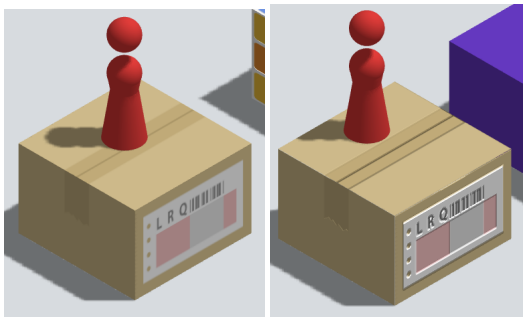

**Figure 6: The effect of bump and displacement mapping**

## 2.5 Lighting affects

We further enhanced the ambient light effect by adding a simulation of sunlight. With this addition, objects appear more three-dimensional and vibrant, making the scene appear more realistic. To achieve this effect, we first added a hemisphere light, implementing a hemispherical shape of ambient light to simulate the sky light and ground reflected light. This light source adds a brighter layer of gray to the sky and a deeper shade of gray to the ground. It is a non-directional light that does not produce shadows but adds more natural light variations to the scene.

We initially enhanced the basic effect of parallel light to simulate sunlight, using an orthographic camera for the shadows. The camera's viewport range and near and far planes are dynamically calculated based on the scene dimensions to ensure the shadows properly cover the required areas. To further simulate a more realistic sunlight effect, we added a directional light source, similar to a spotlight. This light source emits a beam of light from a specific location and renders the scene. The light beam is emitted from the light source and cast at a 60-degree angle onto objects for reflection. We positioned the light source in the upper right corner, acting as the sun, and also adjusted the beam angle to present it at 60 degrees, adding a penumbra effect to make the beam appear softer.

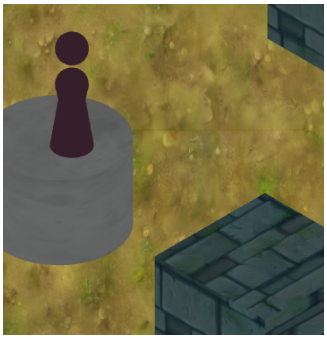

**Figure 7: only ambient light**

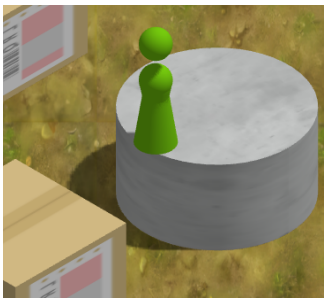

**Figure 8: adding parallel light**

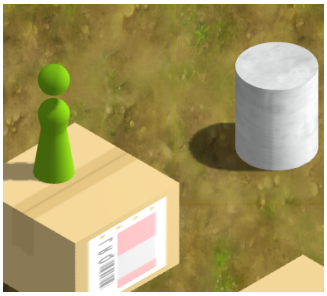

**Figure 9: Adding light source to simulate sunlight**

## 2.6 Different texture maps

At the same time, we also achieved effects with various texture mappings. Previously, when designing the game, we worked against a monochrome background and used predominantly solid-colored blocks. In later designs, we introduced blocks with different textures to enhance the visual appeal of the game. There are four main methods of texture mapping:

The first is background texture mapping, which does not require any coordinate positioning on the texture itself. We simply align the texture's UV coordinates with the x and y coordinates of the scene captured by the camera. If the texture image is too small, you can even repeat some of the texture photos to make the texture mapping blend more naturally with your scene.

The second type involves mapping textures onto rectangular objects. Here, we need to align certain points because the texture image needs to be folded over the object. We start by defining the bottom left corner of the texture image as the origin. Since the

**Figure 10: without texture mapping**

**Figure 11: different texture mapping effects**

**Figure 12: different texture mapping effects**

size of the cube is fixed, the final coverage area is slightly smaller than a 428*428 square. We then locate each vertex's coordinates in the drawn u, v coordinate system and align them with the vertices of the original rectangle to correctly apply the texture.Cylindrical texture mapping is similar but involves only three faces.

The third type of mapping is more complex, designed for imitation. Since a Rubik's cube is not a complete rectangle, we divided it into the top and three side layers for texture mapping. The subsequent operation methods are similar to those used for rectangular texture mapping.

## 3 RESULTS

Ultimately, we successfully created a realistic 'Tiao-Yi-Tiao' minigame. The entire game was modified from a basic code framework, incorporating many different graphical features, making the game more visually appealing and playable. Throughout the coding process, we implemented a variety of graphical knowledge learned in class, including simulation, rotation, texture mapping, as well as different shading and materials effects.

## 4 CONTRIBUTIONS FROM EACH TEAM MEMBER

We worked together to build the basic code and write proposal, milestone and final report.Besides, Yifeng focuses on making particles and trails more interesting and colorful, Sihua focuses on making the littleman's movement more reasonable and physically real, Aslina focuses on lighting, and texture mapping, and Boyuan focuses on shading, materials and making video. All the teammates contribute their best to finding literature and completing assignments, which makes us complete the CS284 final project successfully!

## REFERENCES

[1] https://github.com/yaoshanliang/weapp-jump
[2] https://github.com/wswei99/tiaoyitiao
[3] https://blog.csdn.net/FemaleHacker/article/details/107138639
[4] https://blog.csdn.net/qq45144861/article/details/109709138
[5] https://blog.csdn.net/valada/article/details/79909238

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