Interactive Computer Graphics

Term Project --
3D Mosaic by LEGO™

B89902040 沈允中

B89902094 李沛倫

Abstract

In our childhood, we played a toy like bricks named LEGO™ bricks. It's really interesting and amazing. And in the homework NO.1 and NO.2, we have already learned how to build some models such as CSIE's department building etc. So we think why can we try to build such models by LEGO™ bricks. That is, in this term project, we want to present a method for converting a regular 3-dimensional model into LEGO™ bricks.

Introduction

LEGO™ is a kind of brick. It's 7.8mm in width, 7.8mm in length, and 11.4mm in height (including its head). But unlike regular building blocks, LEGO™ bricks can be assembled together tightly in the top-bottom direction without toppling down. They also come with lots of variety. You can reasonably and easily combine them to form any shape you wand. But the combination is really a NP-hard problem. So in order to simplify this problem, we only use the basic cubic bricks (**Figure. 1**).

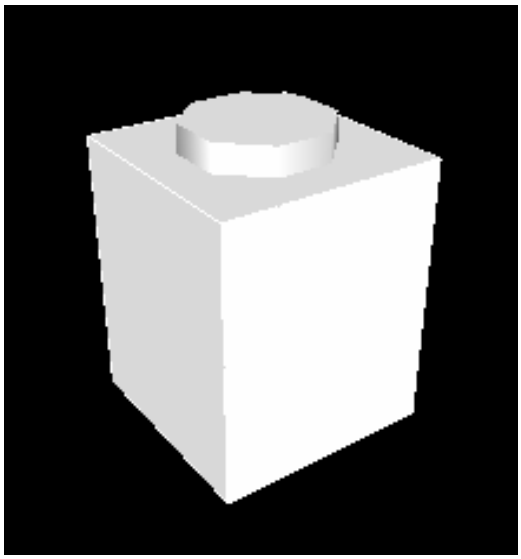

Figure. 1

The Problem

The goal is to truly build our models by really LEGO™ bricks. So we must convert a triangle model into LEGO™ bricks which connected as tightly as possible.

Such problem can be stated formally as follows:

Statement: Given a set of triangles *{T_i}* and an integer *d*, find a set of bricks *{B_j}* such that

*** each *B_j* consists *n_j* by *m_j* unit bricks *{C_k}* of the same color;**

*** each *C_k* is of fixed size and scale, positioned in a grid measured by its size, intersected with some triangles in *{T_i}*, and of the same color with one of the triangle it intersects with; and**

*** *B_j* connects each other as tightly as possible.**

Originally we want to define a function to evaluate how tightly connected the *{B_j}* we found and maximize that function. But in consideration of the efficiency and complexity, we did not implement this method. Instead we use an approximative algorithm which has pretty good results. And we will describe this algorithm below.

Algorithms

The algorithm we use takes two steps. First, the original model is dividing into unit bricks. Second, neighbor bricks of the same color are assembled into bigger bricks as best as possible.

In the first step, the algorithm we use is a little like the inverse of volume rendering's algorithm. We use the technique of Bounding Box. So in dividing the original model, a smallest axis-alighed bounding box contains it is first computed. According the parameter *d*, the box is divided into 8: 10: 8(x: y: z) unit cubes, of which the largest number of unit cubes in one dimension is equal to *d*. Each unit cubes is then calculated whether there is a triangle intersects with it. If so, it is labeled with the color of the triangle.

Let *n* be the number of triangle, *m* be the number of unit cubes. The brute-force approach takes *n x m* times of cube-triangle intersection tests.

To reduce this cost, we divide the problem into *n* one-triangle models. That is to say, we calculate the bounding box (consists unit cubes of the same size) containing each triangle and only the cubes in the bounding box of the triangle is tested. So the average time complexity is *n x* average number of cubes of bounding box of a triangle.

The cube (more correctly, axis-alighed bounding box)-triangle intersection test algorithm consider three cases:

1. if the cube *C_k* contains one of the vertices of the triangle *T_i*, return true;
2. if one of the edges of *C_k* intersects *T_i*, return true;
3. if one of the edges of *T_i* intersects one of the surfaces of *C_k*, return true;

and if all three cases failed, return false.

The neighbor-cubes assembly algorithm is as follows:

1. take a brick *B_j* randomly.
2. take a direction(+x or +z) dir randomly.
3. if the bricks next to *B_j* in the dir direction is connectable(**Figure. 2**), assemble them into one brick.
4. repeat the algorithm several times.

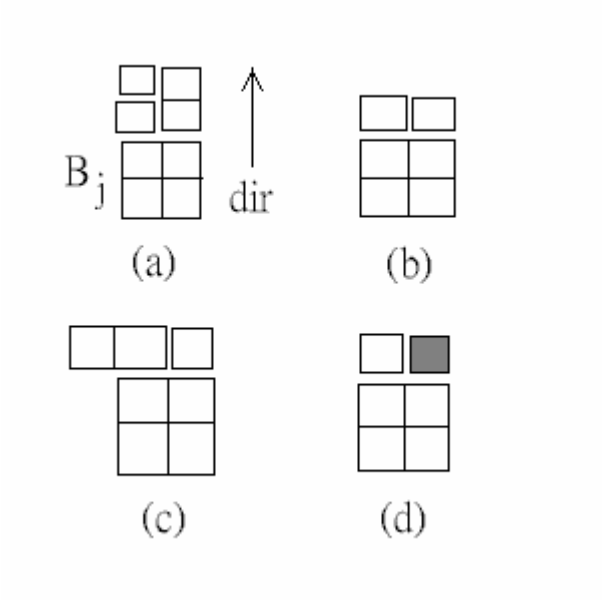

Figure. 2
(a) (b) Cases of bricks that can connect.
(c) (d) Cases of bricks that cannot connect.

As you can see, we did not even test if the set of bricks *{B_j}* is fully-connected, nor did how tightly they connected. There are several reasons:

1. Some models are impossible to be connected tightly at all. For example, The floor of csie model or the wings of fighter model, which are flat surfaces lack of connecting points.
2. The possibility of fully connection for a given model which is possible to be fully-connected is pretty high. And the larger the model, the higher this possibility.

So, consider above reasons, we decide to use the random algorithm currently in use rather than maximize a evaluation function.

Implementation

We implement this project on the OpenGL platform using GLUT library with C language. We also add the animation of the construction and destruction of the bricks to make it fancier.

References

1. In ACM SIGGRAPH 2000(?), AT&T presented a LEGO morphing animation. It is where we got the idea of this project.

Our Results

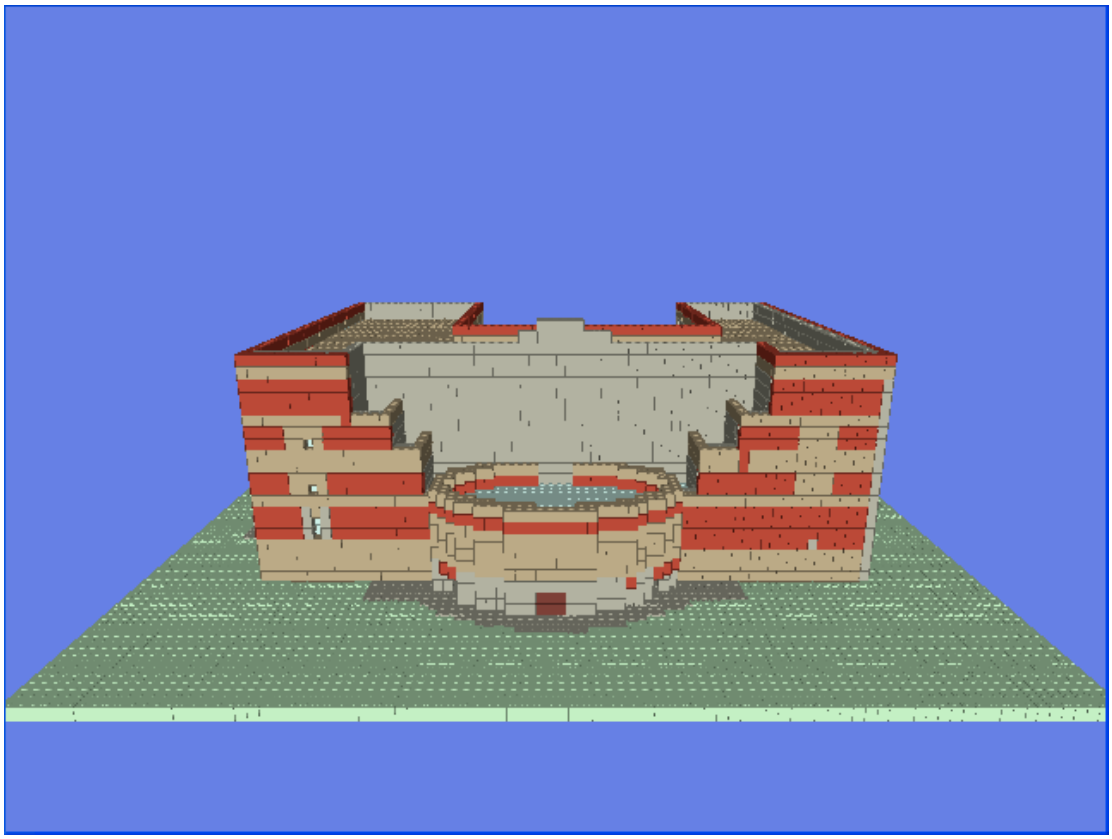

Figure. 3 csie.tri

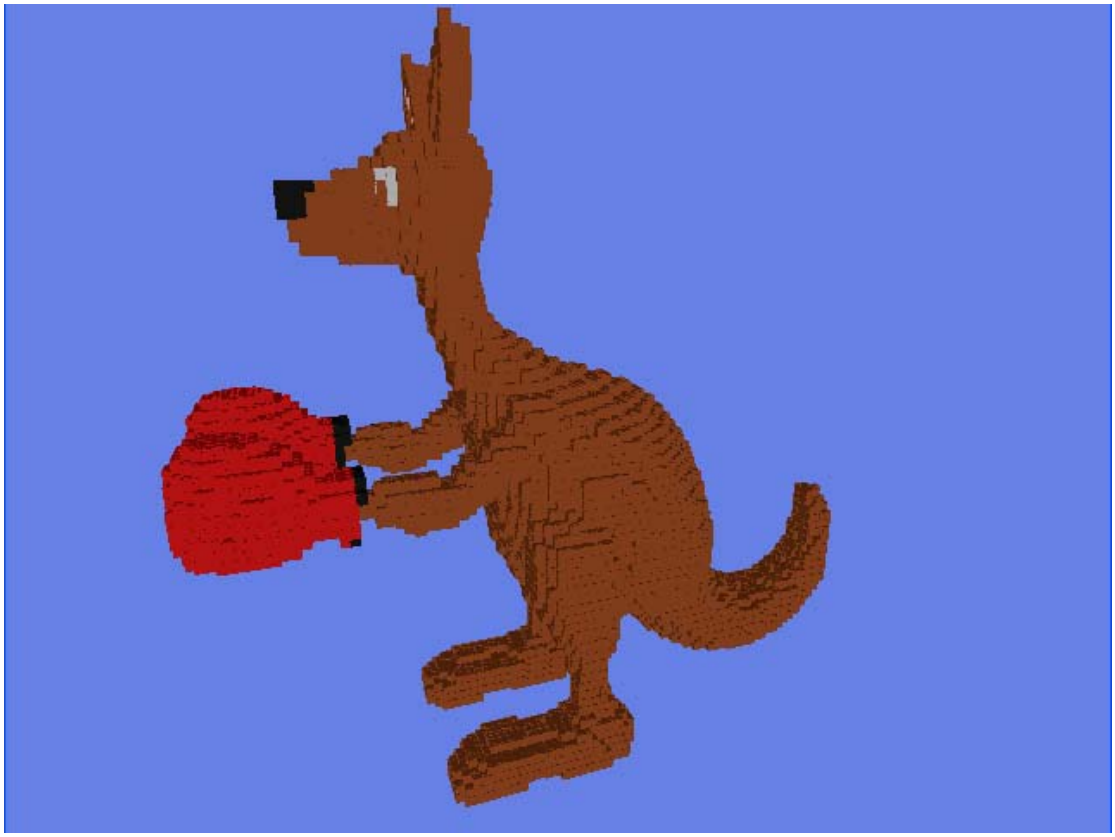

Figure. 4 kangaroo.tri

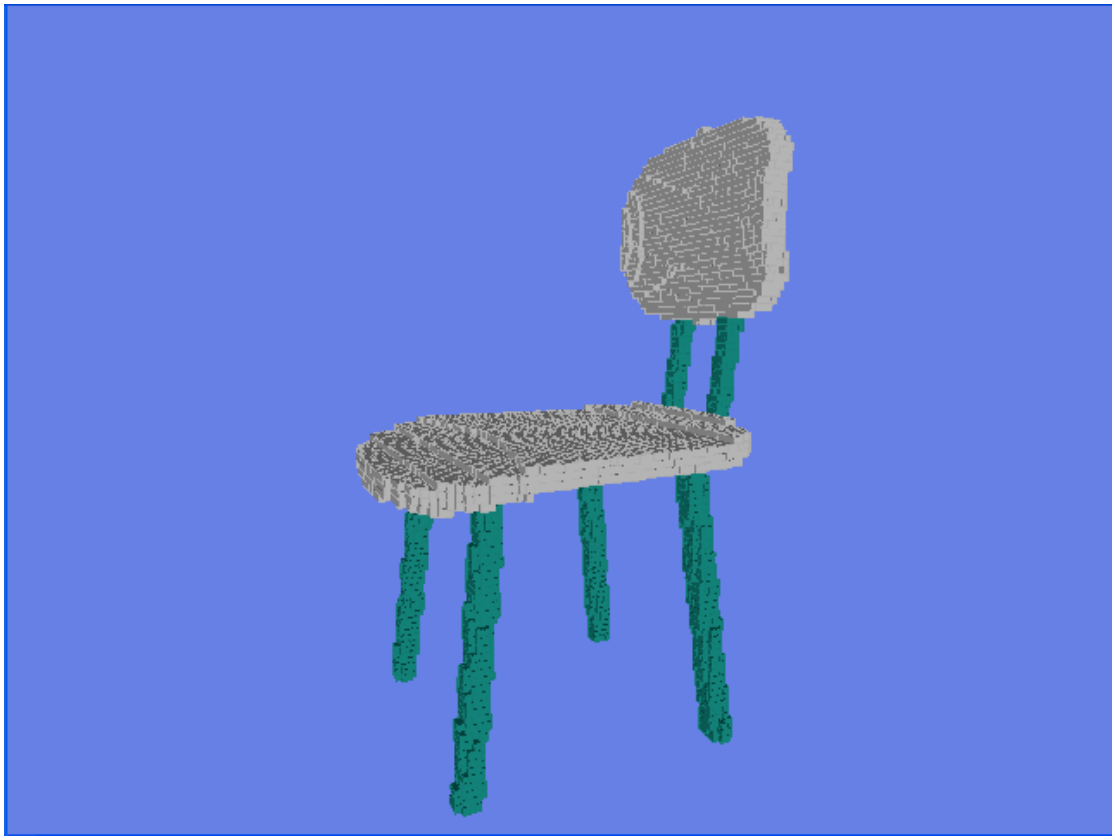

Figure. 5 patchair.tri

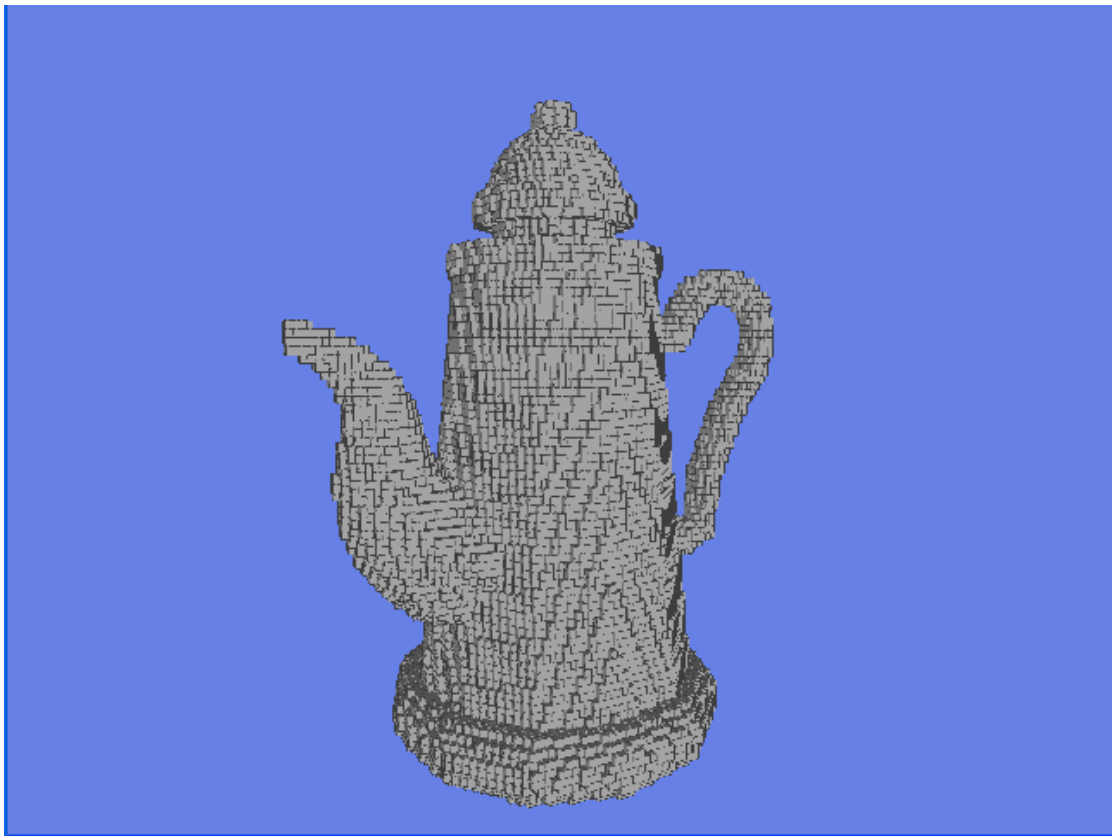

Figure. 6 teapot.tri

Conclusion And Future Work

We have presented a method to convert a triangle model into descrete LEGO™ bricks. The method is efficient while easy to implement. And the result is good enough to our expectation.

But there are still several directions to improve. First, the overall structure of the bricks is correct but not perfect. In human's view, sometimes it needs to be added or removes some of the bricks or modify the color to make it more symmetric. Second, as discussed in previous section, we can introduce the evaluation function for a configuration, for which further study is needed.