THE COLLEGE OF WOOSTER

Marching Cubes 3D Isosurface Rendering Algorithm

Algorithm Analysis Independent Study

Author:

Jianqiu Bai

Advisor:

Dr. Denise Byrnes

Abstract

The Marching Cubes (MC) algorithm is the most commonly used approach for 3D rendering. The method is described in [12], by Lorensen and Cline. The basic idea is extracting triangles from scalar volumetric data sets, and then forming all triangles into an isosurface [10]. First of all, this paper introduces the concept of isosurface and volumetric data. Then, the paper introduces the algorithm by addressing a sample process: rendering a rectangular pyramid from given data. Next, the paper shows a implementation procedure for a MC algorithm visualization tool, in C++ through Visual Studio or Xcode. In the end, the paper states the limitations of the MC algorithm and the future work of the program.

Keyword: Marching Cubes, isosurface, volumetric data, render.

Contents

List of Figures

1 Introduction

With the development of medicine and computer science, people have begun to combine the modern techniques between these two fields. For instance, the Marching Cubes (MC) algorithm renders a 3D isosurface through a computer from a data set captured by the medical machines like computerized tomography (CT) scanners or magnetic resonance imaging (MRI) scanners [10]. Figure 1 below shows a sample work flow, in which the MC algorithm is utilized to render a human brain to the right of Figure 1 from a data set corresponding to the MRI images to the left of Figure 1.

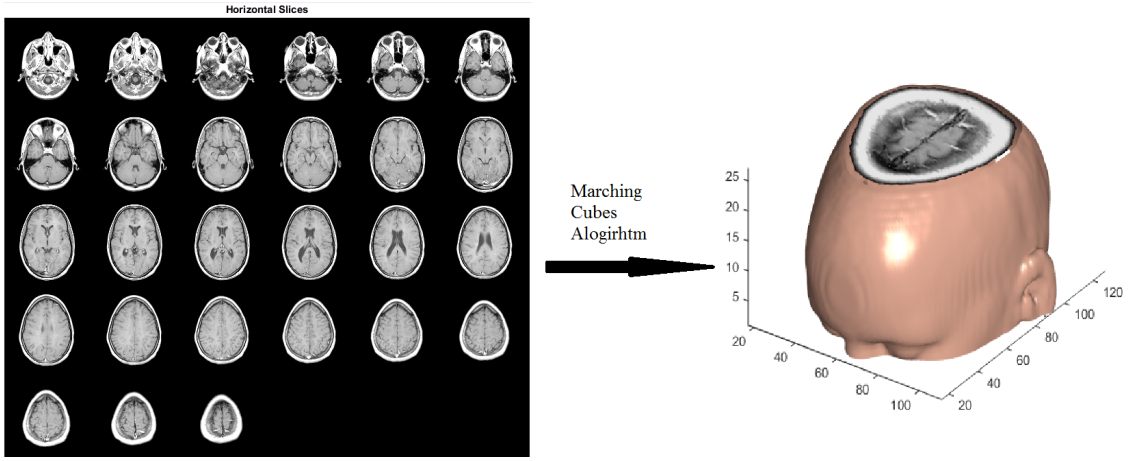


Figure 1: Sample Work Flow of MC Algorithm [5].

The advantage of this procedure is that physicians are able to have a 3D visualization on the rendered model, instead of viewing 2D MRI images. This helps physicians to better analyze the patients' condition. This paper will not go into detail about how CT or MRI scanners capture data, but focus on how the MC algorithm renders the isosurface from certain given data sets. To make readers have a better understanding of the algorithm, some concepts are first defined in the following subsections, and we will then process rendering a rectangular pyramid from its corresponding data set to introduce this algorithm in the next section.

1.1 Isosurface

Recall that the target of the MC algorithm is an isosurface. We first introduce two definitions, shown below, of an isosurface.

Definition 1. Isosurface - is defined as a scalar field $F(P) : \mathbb{R}^3 \to \mathbb{R}$, where P is a 3D point, (x, y, z) [10].
Definition 2. Isovalue - is a constant denoted as α where $F(P) = \alpha$ [10].

In other words, an isosurface refers to a set of 3D points with the same isovalue. By definition, one can see that an isosurface is a surface which is consistent [10]. In the real world, people can regard isosurface as any surface that has the same density everywhere, or the surface of any object that is made of the same material, like human tissues or bones. For example, the surface of the teeth in the Figure 2(c), or the surface of the machine material in Figure 2(d), is an isosurface. In mathematics, the term 'isosurface' is more implicit. We can call a rectangular pyramid or a ring an isosurface, as long as it is constructed by finite points where each point has the same scalar value [10], like Figure 2(a) and Figure 2(b).

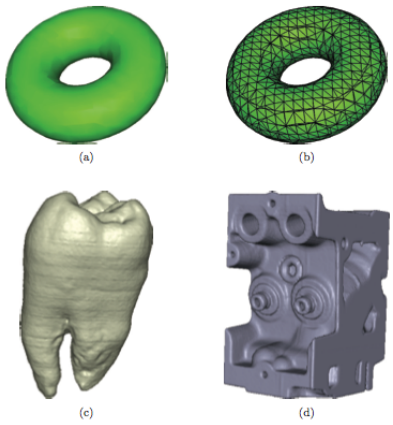


Figure 2: Examples of Isosurfaces [10].

1.2 Volumetric Data

To render an isosurface, the algorithm takes volumetric data as input, and the data is usually represented as a regular 3D grid. Hence, we have following definitions.

Definition 3. Regular 3D grid - is a partition of a large cuboid in $\mathbb{R}^3$ into small congruent cubes (see Figure 3) [10].
Definition 4. Grid Cube - is a small congruent cube in a regular 3D grid [10].
Definition 5. Grid Vertex - is a vertex on a grid cube.
Definition 6. Grid Edge - is an edge on a grid cube.

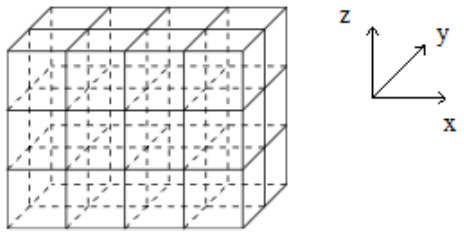


Figure 3: A Regular 3D Grid $(4 \times 2 \times 3)$ [10].

For consistency, we label the coordinates direction like the one in Figure 3 And the size of a regular 3D grid is denoted as $x \times y \times z$, which means there are x cubes along the $x-$axis, y cubes along the $y-$axis and z cubes along the $z-$axis. The definition of volumetric data follows as below.

Definition 7. Volumetric Data - is a set of grid vertex P mentioned above, where each grid vertex has a value to represent its property [8].

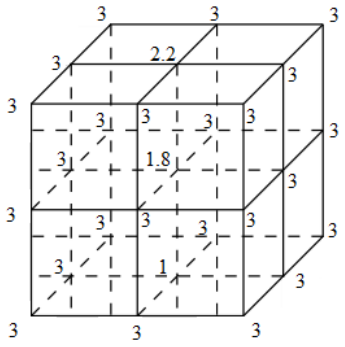
The Figure 4(a) shows the given volumetric data of the rectangular pyramid that we are rendering. In this data set, each value refers to a scalar value of a grid vertex, and the isovalue for this given data set is 2. Since the size of the 3D grid is $x \times y \times z$ in general, the size of the data is $(x+1) \times (y+1) \times (z+1)$. Based on the size of the data, each value can be implicitly set into the corresponding 3D grid like Figure 4(b), and the implementation to assign the value from volumetric data to a 3D grid is explained in the next section.

```
3, 3, 3,
3, 1, 3,
3, 3, 3,

3, 3, 3,
3, 1.8, 3,
3, 3, 3,

3, 3, 3,
3, 2.2, 3,
3, 3, 3
```

(a) Volumetric Data Sample $(3 \times 3 \times 3)$ with isovalue of 2.



(b) 3D Grid Sample $(2 \times 2 \times 2)$.

Figure 4: Volumetric Data Samples.

One important thing for the audience to notice is that, the actual volumetric data is stored like Figure 4(a). For people to better understand the algorithm, the original data are implicitly stored like 4(b). Here, one can see that the direction to assign the volumetric data to the 3D grid does not really matter, as long as all volumetric data are assigned in order. For consistency, we assign the volumetric data along with the x direction first, then along with the y direction and finally along with the z direction.

2 Algorithm

After understanding how volumetric data is recorded, we then are able to see how the MC algorithm works. For a simpler explanation to our audience, we keep progressing on a rectangular pyramid rendering through the whole section. Based on Figure 4(b), the expected result should look like Figure 5. The algorithm can be divided into three steps [10], and the following subsections explain each step.

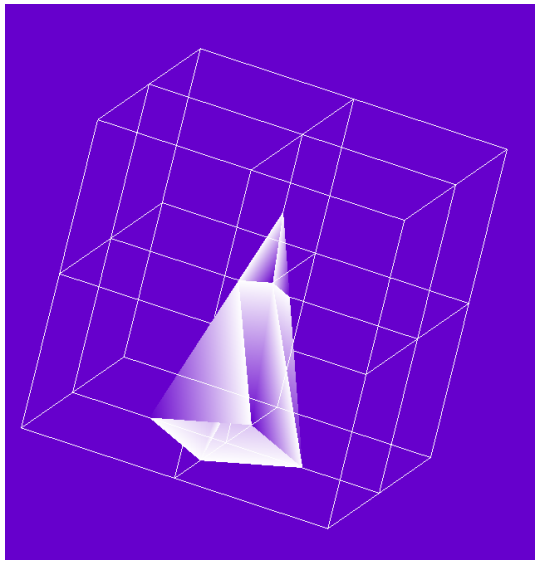


Figure 5: The Expected Result of the Rectangular Pyramid.

2.1 Step 1 - Read Isosurface Lookup Table

According to our expected graph (see Figure 5), one can see that, in this imaginary 3D grid, some grid vertexes are on or inside the isosurface, and others are outside. Recall that, from the previous section, we have assigned the volumetric data to the corresponding 3D grid (from Figure 4(a) to Figure 4(b)). Imagine we take out

the closest top-left grid cube from Figure 4(b), we should obtain a cube like Figure 6, where the numbers inside the parentheses are values from the volumetric data. For consistency again, we label the eight grid vertexes shown in Figure 6, and we label every grid vertexes for each grid cube in the same way.

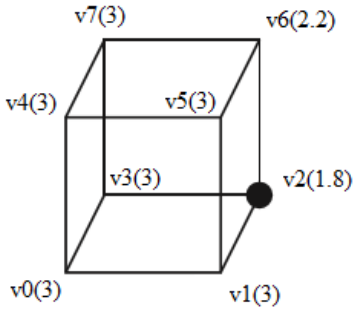


Figure 6: A Sample Cube with Value to Each Labeled Grid Vertex [10].

Comparing the value to each grid vertex with the given isovalue, we can determine a grid vertex is on or inside the isosurface if the gird vertex value is equal to or less than the isovalue. Otherwise, the grid vertex is outside the isosurface. In Figure 6, one can see that only gird vertex v_2 for this grid cube sits inside the isosurface, all other grid vertexes are outside the isosurface. In general, for each grid cube, there are $2^8 = 256$ cases, and we have the following definition.

Definifion 9. Vertex Configuration - is the configuration of vertex labels for each grid cube [10].

In fact, for each vertex configuration, there is only one corresponding set of triangle edges, and is defined below.

Definition 10. Edge Configuration - is the configuration of edge labels associated to the vertex configuration [10].

Figure 7 shows the edge configuration matched to Figure 6. Again, we label the twelve grid edges in the way shown in Figure 7 for each grid cube. In Figure 7, one can see that this grid cube only contains one triangle, and this type of triangle is defined as below.

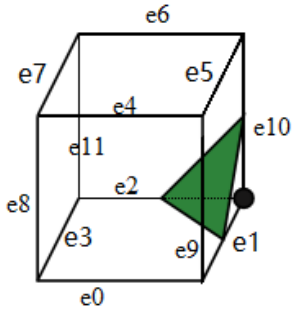


Figure 7: A Sample Cube Containing Constructional Triangles with Labeled Grid Edge [10].

Definition 11. Constructional Triangle - is a piece of triangle as part of an isosurface [10].

Combining all constructional triangles, we can then render the isosurface, like Figure 8 shows below, which is the basic idea of the MC algorithm [11].

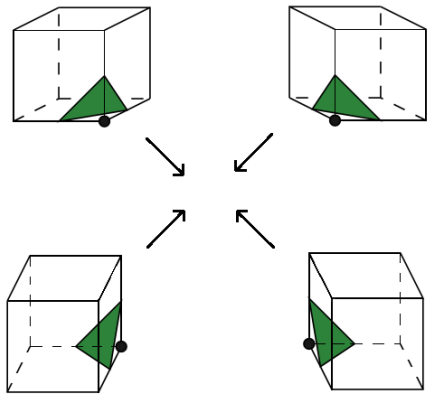


Figure 8: A Sample Combing All Constructional Triangles in the Top Gird Cubes [10].

Lorensen and Cline concluded that there are all 256 edge configurations, and made a Lookup Table as a visual representation of all edge configurations [8]. Since some configurations are rotations or reflections of one another, the 256 configurations can be reduced to 22 distinct configurations [10] (see Appendix1 and Appendix2).

Above all, the first step actually constructs a 3D grid by the given volumetric data, goes through each grid cube to find its vertex configuration, and reads

the corresponding edge configuration containing constructional triangles from the Lookup Table.

2.2 Step 2 - Retrieve Isosurface Triangle Edges for Each Grid Cube

From the edge configuration, we then are able to retrieve the isosurface triangle edges if there is one. As Figure 7 shows, three vertexes of this constructional triangle sit on the grid edge e_1, e_2 and e_{10}. Then, we have the following definitions.

Definition 12. Isosurface Vertex - is the vertex of the constructional triangles.
Definition 13. Isosurface Triangle Edge - is a grid edge which contains an isosurface vertex [10].

Thus, for the edge configuration in Figure 7, e_1, e_2 and e_{10} are isosurface triangle edges for the current grid cube.

2.3 Step 3 - Compute Isosurface Vertex Coordinates

After retrieving the isosurface triangle edges for each grid cube, we need to compute the coordinates of the isosurface vertex. We use linear interpolation in Mathematics for this computation by the formula

$$v = (1 - k)P_p + P_q,$$

where $k = \frac{\alpha - V_p}{V_q - V_p}$, P_p, P_q are the coordinates of the two endpoints of an isosurface triangle edge, and V_p, V_q are vertex values at P_p, P_q.

For example, from Figure 7, we have retrieved isosurface trianlge edges e_1, e_2 and e_{10}. Combining the vertex configuration from Figure 6 with the edge configuration from Figure 7, we obtain Figure 9 shown below.

By linear interpolation, we can obtain the coordinates of the isosurface vertexes (labeled as red, yellow and blue vertexes). After going through each grid cube in the imaginary 3D grid, one is able to get all the isosurface vertexes, which are the output for the MC algorithm.

3 Program Implementation

After learning about the MC algorithm, this section shows how to build a MC algorithm visualization tool, by the MC algorithm and the C++ GLUT. The C++

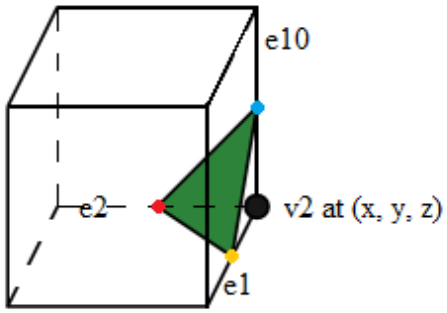


Figure 9: A Sample Grid Cube Combing where the Red, Yellow and Blue Vertexes Are Isosuarface Vertexes [10].

GLUT is the OpenGL Utility Toolkit [3], where OpenGL is a library extension for rendering [4].

The program tool stores the volumetric data for five different models. The GUI (Graphics User Interface) is developed in this tool (see Figure 10). Users can right-click the mouse in the window to open the menu and select the model to render. Instructions, rotations and color display modes are implemented in the program tool as well. Once a model is chosen, the program runs the MC algorithm inside to access all isosurface vertexes of the model, and the GLUT links all the isosurface vertexes and displays all constructional triangles to render the model.

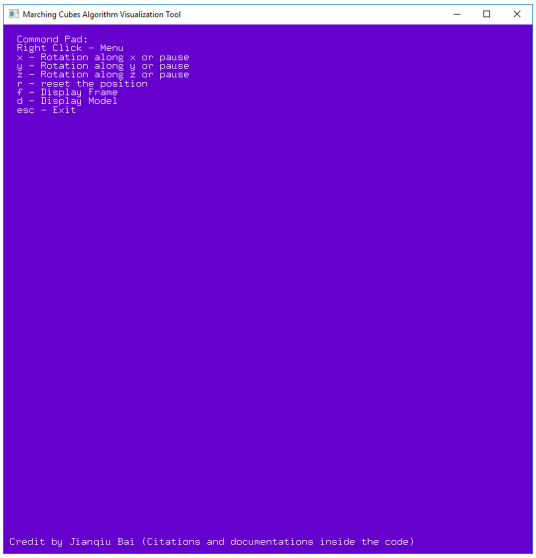


Figure 10: The GUI of the MC Algorithm Visualization Tool

3.1 Models

The volumetric data of the five models are stored in the program and shown in the table below.

Model	Size	Isovalue	Type
Octahedron	$3 \times 3 \times 3$	2	Regular Isosurface
Rectangular Pyramid	$3 \times 3 \times 3$	2	Regular Isosurface
Sphere	$21 \times 21 \times 21$	100	Regular Isosurface
Flow [6]	$50 \times 25 \times 50$	-3	Irregular Isosurface
Brain [5]	$128 \times 128 \times 27$	20	Irregular Isosurface

3.2 Data Structure

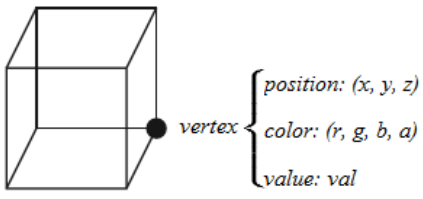
Before implementing the MC algorithm, the next task is to declare a class object called $vertex$ in this program, to hold and access the information for the grid vertexes and the isosurface vertexes. Based on the algorithm, we shall see that both grid vertexes and isosurface vertexes should contain the coordinates and the value. To make color coding in the implementation, each vertex should also contain the RGBA value [9]. Therefore, a class $vertex$ has three private element: $position$, val and $color$.

The $position$ is a $struct$ data type $vec3$ holding coordinates (x, y, z) of a vertex. The $color$ is a $struct$ data type $vec4$ holding RGBA values (r, g, b, a) of a vertex. The val is a $float$ containing the value to a vertex. Figure 11 shows the $vertex$ data structure.

```
typedef struct
{
    GLfloat x, y, z;
} vec3;

typedef struct
{
    GLfloat r, g, b, a;
} vec4;
```

(a) The Codes for the $vec3$ and $vec4$ Struct Data Type



(b) Data Structure Shown in a Grid Cube

Figure 11: The $vertex$ Data Structure Sample

All types of constructor, accessor, modifier and display functions are implemented in the $vertex$ class, so one can easily get and change (x, y, z), (r, g, b, a) or the value to any vertex.

3.3 MC Implementation

Once the basic data structure is constructed, we are able to implement the MC algorithm. Based on the three steps mentioned in the previous section, the procedure of the MC implementation is:

1. Assigning values from the given volumetric data to each grid vertex in the imaginary 3D grid mentioned in Section 1.2.
2. Retrieving the Lookup Table.
3. Reading the Lookup Table (step 1).
4. Retrieving isosurface triangle edges (step 2).
5. Using linear interpolation to compute the isosurface vertexes (step 3).

3.3.1 Assgining Values

We first use $vector\langle float\rangle$ data structure called $inputData$ to hold all the values from the volumetric data. We use a $triple - loop$ in C++ shown in Figure 12, to go through each grid cube in the 3D grid. We use the name $maxX$ referring to x, $maxY$ referring to y and $maxZ$ referring to z.

```
for (GLint k = 0; k < maxZ; k++)
{
        for (GLint j = 0; j < maxY; j++)
        {
                for (GLint i = 0; i < maxX; i++)
                {
                        //get the pos of all 8 vertices for each cube
                        vertex v_0(i, j, k, 0);
                        vertex v_1(i, j + 1, k, 0);
                        vertex v_2(i + 1, j + 1, k, 0);
                        vertex v_3(i + 1, j, k, 0);
                        vertex v_4(i, j, k + 1, 0);
                        vertex v_5(i, j + 1, k + 1, 0);
                        vertex v_6(i + 1, j + 1, k + 1, 0);
                        vertex v_7(i + 1, j, k + 1, 0);
```

Figure 12: The Code of the Triple-loop to Go Through All Eight Grid Vertexes for Each Grid Cube

Next, for each iteration in the $loop$, the algorithm assigns the value from the $vector\langle float\rangle$ $input$ to its corresponding grid vertex. Recall the volumetric data of the rectangular pyramid shown in Figure 13(a) below, the size of this data set is $2 \times 2 \times 2$. Therefore, Figure 13(b) shows the associated 3D grid containing

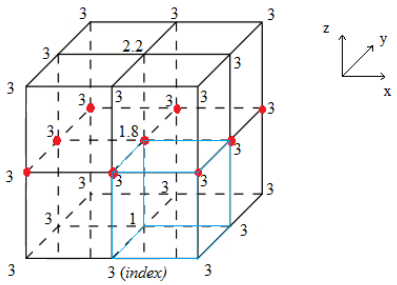
$(maxX + 1)(maxY + 1)(maxZ + 1) = (2 + 1)^3 = 27$ grid vertexes,
$(maxZ + 1) = 2 + 1 = 3$ layers, and
$(maxX + 1)(maxY + 1) = 9$ grid vertexes in each layer.

```
3, 3, 3,
3, 1, 3,
3, 3, 3,

3, 3, 3,
3, 1.8, 3,
3, 3, 3,

3, 3, 3,
3, 2.2, 3,
3, 3, 3
```

(a) $vector\langle float\rangle\ inputData$.



(b) 3D Grid Sample.

Figure 13: Volumetric Data Samples.

Assume it is the 2nd iteration, so we are looking at the 2nd gird cube, the closest bottom-right one (colored as blue) in Figure 13(b). For this current grid cube, the volumetric data values are labeled with blue underlines in Figure 13(a). Figure 14 shows how we access the data.

```
GLint XY = (maxX + 1)*(maxY + 1);
GLint index = i + (maxX + 1)*j + XY*k;

//assign the value to all 8 vertices for each cube
v_0.changeVal(inputData[index]);
v_1.changeVal(inputData[index + (maxX + 1)]);
v_2.changeVal(inputData[index + (maxX + 1) + 1]);
v_3.changeVal(inputData[index + 1]);
v_4.changeVal(inputData[index + XY]);
v_5.changeVal(inputData[index + (maxX + 1) + XY]);
v_6.changeVal(inputData[index + (maxX + 1) + XY + 1]);
v_7.changeVal(inputData[index + XY + 1]);
```

Figure 14: The Code of Assigning Values from the Volumetric Data to the Grid Vertex.

The variable XY contains the numbers of the grid vertexes in each layer. The variable $index$ indicates the index of the first grid vertex v_0 (the closest bottom-left one) for each grid cube in $inputData$. According to Figure 13, for each grid cube, four gird vertexes are in one layer, and the other four vertexes are in the layer above. Therefore, we have the indexes for all eight grid vertexes for each grid cube shown in Figure 14.

3.3.2 Retrieving the Lookup Table

Next, in order to read the Lookup Table for step 1, we need to store the Lookup Table in the program. Recall that the Lookup Table consists of graphical edge configurations, we need to convert the graphical configurations into numerical configurations for program implementation. Since there are a finite 256 configurations, Paul labeled all configurations, and generated a Lookup Array (see Appendix3) and a Lookup Matrix (see Appendix4) [2]. The Lookup Array is for reading the Lookup Table (step 1). The Lookup Matrix is for retrieving isosurface edges (step 2). Figure 15 shows the transformation of reading a graphical configuration in the Lookup Array, and retrieving the isosurface edges in the Lookup Matrix.

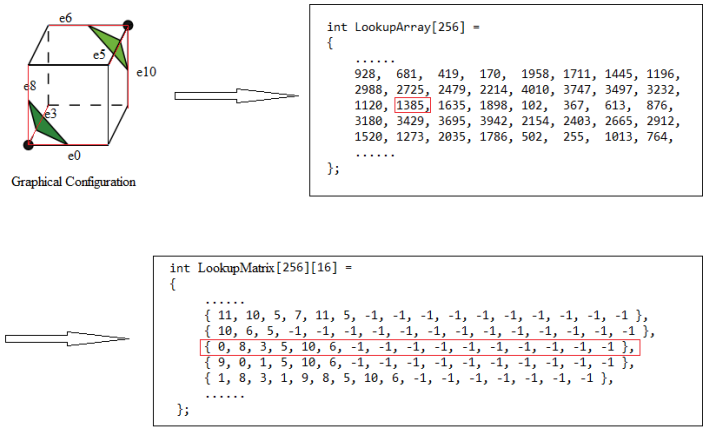


Figure 15: The Sample Transformation [2] [10].

The Lookup Array contains 256 numbers. Each number implicitly refers to each edge configuration. For example, the grid cube in Figure 15 has 6 isosurface triangle edges as $\{e_0, e_3, e_8, e_5, e_6, e_{10}\}$. We use a binary representation to indicate an isosurface triangle edge:

$$e_0 - 0000\ 0000\ 0001 = 1,\ e_3 - 0000\ 0000\ 1000 = 8,\ e_8 - 0001\ 0000\ 0000 = 256,$$
$$e_5 - 0000\ 0010\ 0000 = 32,\ e_6 - 0000\ 0100\ 0000 = 64,\ e_{10} - 0100\ 0000\ 0001 = 1024.$$

Hence, $1+8+256+32+64+1024 = 1385$, which refers to the edge configuration $\{e_0, e_3, e_8, e_5, e_6, e_{10}\}$.

The Lookup Matrix has the size of 256×16, where each row represents each edge configuration as well. Based on the graphical Lookup Table, one can see that a grid cube can contain five constructional triangles at most. Thus, there are $5 \times 3 = 15$ grid edges at most. In the Lookup Matrix, Paul used integers from 0 to 11

to represent the grid edge that is an isosurface trianlge edge, and -1 to represent the other grid edges [2]. Therefore, there are 15 elements in each row, and another -1 is added as the ending for each row. Hence, the size of the Lookup Matrix is 256×16.

3.3.3 Read Lookup Table (Step 1)

Based on Figure 15, one can see that reading Lookup Table means finding the number in the Lookup Array that the current edge configuration is referring to. According to the algorithm in step 1, we need to compare the value to each grid vertex with the given isovalue, to first obtain a vertex configuration. In fact, the number from 0 to 255 implicitly refers to each vertex configuration. To find the number for the current vertex configuration, we use bit-wise addition operation shown in Figure 16 [1].

```
//check if each vertex is inside or outside
GLint inOutIndex = 0;
if (v_0.getVal() <= isoVal) (inOutIndex |= 1); //0000 0001
if (v_1.getVal() <= isoVal) (inOutIndex |= 2); //0000 0010
if (v_2.getVal() <= isoVal) (inOutIndex |= 4); //0000 0100
if (v_3.getVal() <= isoVal) (inOutIndex |= 8); //0000 1000
if (v_4.getVal() <= isoVal) (inOutIndex |= 16); //0001 0000
if (v_5.getVal() <= isoVal) (inOutIndex |= 32); //0010 0000
if (v_6.getVal() <= isoVal) (inOutIndex |= 64); //0100 0000
if (v_7.getVal() <= isoVal) (inOutIndex |= 128); //1000 0000
```

Figure 16: The Code for Computing the Number to Match the Lookup Array.

3.3.4 Retrieving Isosurface Triangle Edges & Linear Interpolation Computation (Step 2 & Step 3)

After retrieving the index variable $inOutIndex$ in Figure 16, one can read the associated edge configuration by calling $LookupArray[inOutIndex]$. Recall that $LookupArray[inOutIndex]$ is a number like 1385 in Figure 15, our next step is to track how this number is added by bit-wise 'and' operation shown in Figure 17 [1]. Since $1385 = 1 + 8 + 256 + 32 + 64 + 1024$, for this grid cube, the if-statements are true in lines 2, 5, 7, 8, 10 and 12 in Figure 17. Thus, the linear interpolation is computed on the grid edges $e_0, e_3, e_5, e_6, e_8, e_{10}$, where each grid edge has a pair of endpoints of $(v_0, v_1), (v_0, v_3), (v_5, v_6), (v_6, v_7), (v_0, v_4)$ and (v_2, v_6) (see Figure 6 and Figure 7).

```
vertex intersectV[12];
if (LookUpArray[inOutIndex] & 1) intersectV[0] = linearInterpolation(v_0, v_1, isoVal); //0000 0000 0001
if (LookUpArray[inOutIndex] & 2) intersectV[1] = linearInterpolation(v_1, v_2, isoVal); //0000 0000 0010
if (LookUpArray[inOutIndex] & 4) intersectV[2] = linearInterpolation(v_3, v_2, isoVal); //0000 0000 0100
if (LookUpArray[inOutIndex] & 8) intersectV[3] = linearInterpolation(v_0, v_3, isoVal); //0000 0000 1000
if (LookUpArray[inOutIndex] & 16) intersectV[4] = linearInterpolation(v_4, v_5, isoVal); //0000 0001 0000
if (LookUpArray[inOutIndex] & 32) intersectV[5] = linearInterpolation(v_5, v_6, isoVal); //0000 0010 0000
if (LookUpArray[inOutIndex] & 64) intersectV[6] = linearInterpolation(v_7, v_6, isoVal); //0000 0100 0000
if (LookUpArray[inOutIndex] & 128) intersectV[7] = linearInterpolation(v_4, v_7, isoVal); //0000 1000 0000
if (LookUpArray[inOutIndex] & 256) intersectV[8] = linearInterpolation(v_0, v_4, isoVal); //0001 0000 0000
if (LookUpArray[inOutIndex] & 512) intersectV[9] = linearInterpolation(v_1, v_5, isoVal); //0010 0000 0000
if (LookUpArray[inOutIndex] & 1028) intersectV[10] = linearInterpolation(v_2, v_6, isoVal); //0100 0000 0000
if (LookUpArray[inOutIndex] & 2056)  intersectV[11] = linearInterpolation(v_3, v_7, isoVal); //1000 0000 0000
```

Figure 17: The Code for Retrieving the Isosurface Triangle Edges and Linear Interpolation Computation.

Then, for certain indexes of the array $intersectV$ in Figure 17, it should store the isosurface vertexes. For instance, if $LookupArray[inOutIndex]$ is 1385, then the $intersectV$ should be

Index	0	1	2	3	4	5	6	7	8	9	10	11
Isosurface Vertex	Yes	No	No	Yes	No	Yes	Yes	No	Yes	No	Yes	No

In the end, we pick up all the isosurface vertexes, and store into a triangle list shown in Figure 18

```
for (GLint n = 0; edgeTripleLookUp[inOutIndex][n] != -1; n += 3)
    {
        tri.v[0] = intersectV[edgeTripleLookUp[inOutIndex][n]];
        tri.v[1] = intersectV[edgeTripleLookUp[inOutIndex][n + 1]];
        tri.v[2] = intersectV[edgeTripleLookUp[inOutIndex][n + 2]];
        triLst.push_back(tri);
    }
```

Figure 18: The Code For Storing the Isosurface Vertexes into a Triangle List.

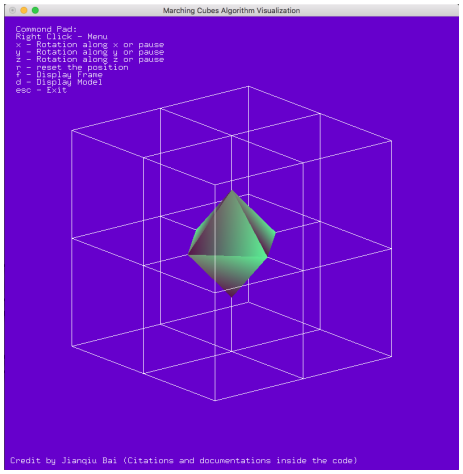
3.4 OpenGL Implementation

After running the MC algorithm, we obtain a list of (x, y, z) values in order, for all isosurface vertexes. Then, our task is using OpenGL to render the isosurface vertexes. We use GLUT Tookit from OpenGL and implement the following four parts [7]:

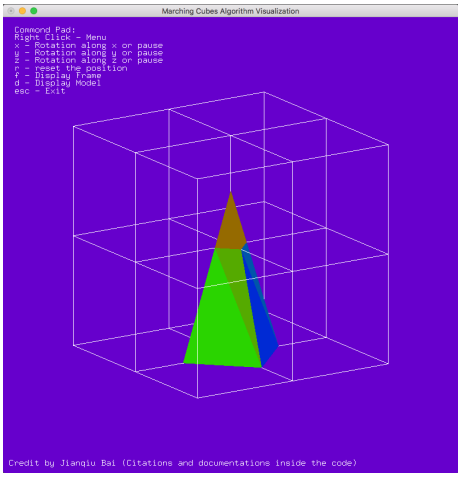
1. Fundamental OpenGL functions.
2. Color mode - default color, random color, depth color, shade color and transparency.
3. User interface - instruction display and menu.
4. User controller - rotations and frame display.

4 Results

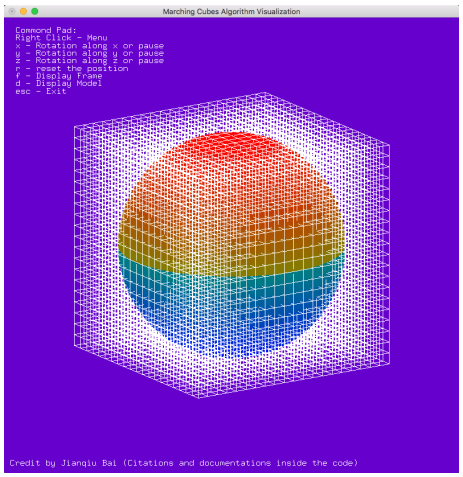
In this section, some results of running the program are shown below.



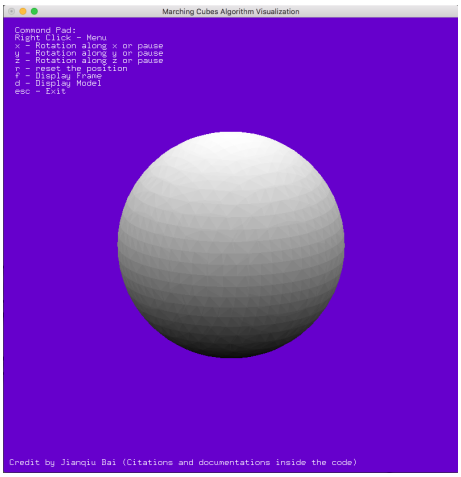
(a) The Octahedron Model with Random Color and Frame



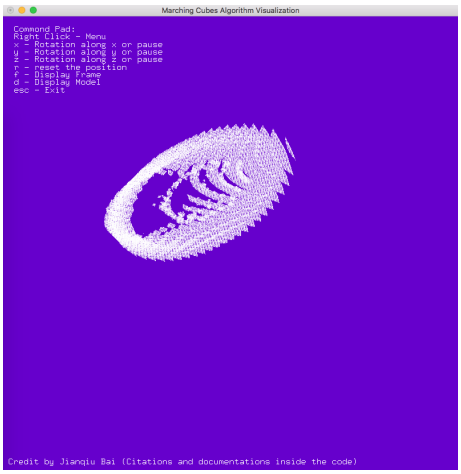
(b) The Rectangular Pyramid Model with Frame.



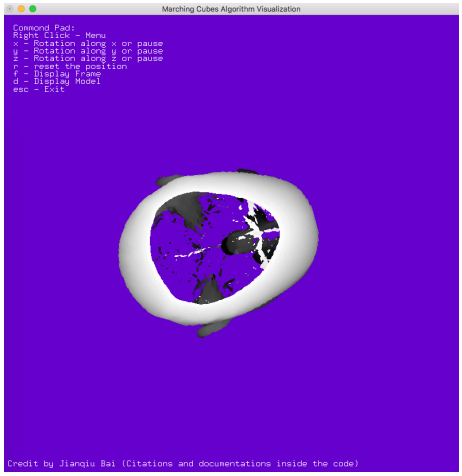
(a) The Sphere Model with Frame.



(b) The Sphere Model with Shade Color.



(a) The Transparency Flow Model.



(b) The Brain Model with Shade Color.

Figure 19: Sample Results.

5 Conclusion

In this paper, we have shown the MC algorithm by rendering a rectangular pyramid as an example. Then, we have implemented a MC visualization tool consisting of five models.

According to the algorithm, since we use a $triple - loop$ to go through each grid cube, the total run time simply yields to $\Theta(n^3)$. Thus, for a large volumetric data set, the MC algorithm might be running slowly, which is a limitation of this algorithm.

For the program, on can see tee are only five fixed models, In the future, we hope to implement an 'add file' function. Thus, users are able to add any volumetric data into the program to render any model they want.

References

[1] ALLEN, A. Bitwise operators in c and c++. http://www.cprogramming.com/tutorial/bitwise_operators.html.

[2] BOURKE, P. http://paulbourke.net/geometry/polygonise/, 1994.

[3] KHRONOS. http://paulbourke.net/geometry/polygonise/.

[4] KHRONOS. https://www.opengl.org/.

[5] MATHWORKS. Exploring slices from a 3-dimensional mri data set. http://paulbourke.net/geometry/polygonise/, 2017.

[6] MATHWORKS. Isosurface. https://www.mathworks.com/help/matlab/ref/isosurface.html, 2017.

[7] STUART, F. R. *Practical Algorithms for 3D Computer Graphics*. A K Peters/CRC Press, 2013, ch. Chapter 7, pp. 263 – 292.

[8] TIMOTHY S. NEWMAN, H. Y. A survey of the marching cubes algorithm. *Computer Graphics 30* (2006), 854–879.

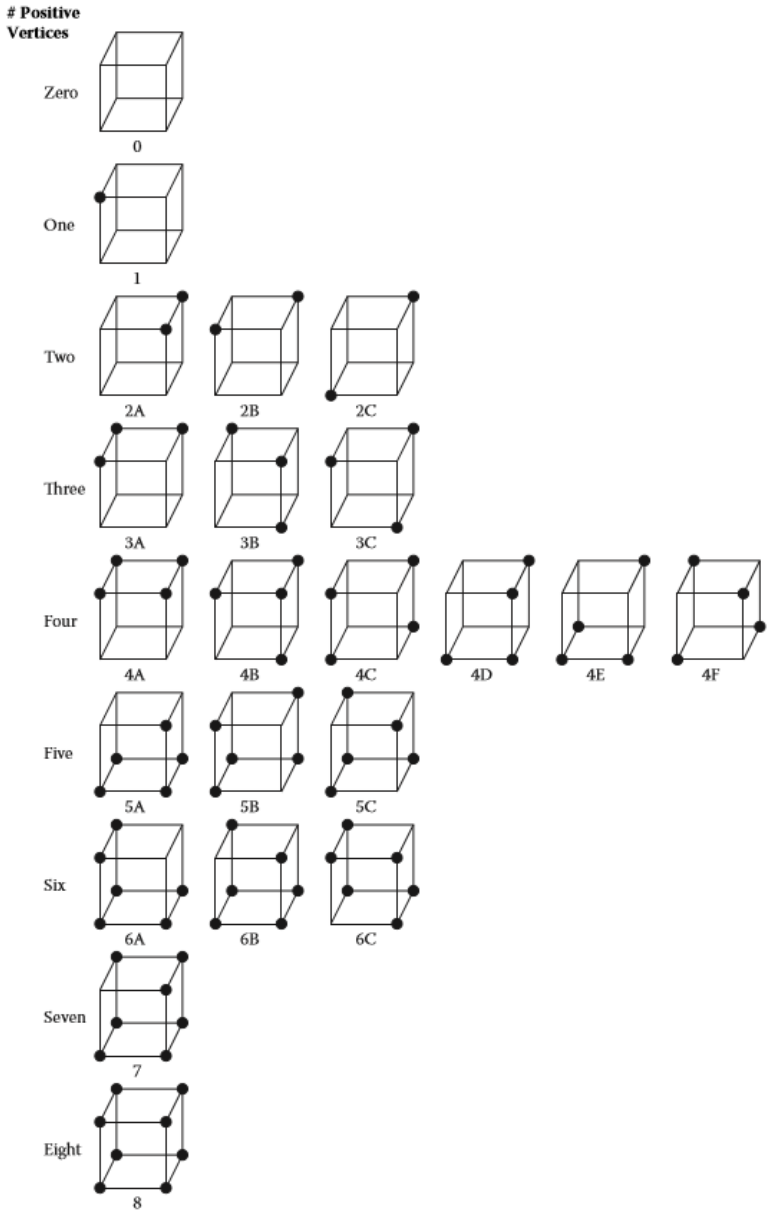
[9] W3SCHOOLS. Rgba colors. https://www.w3schools.com/css/css3_colors.asp, 2017.

[10] WENGER, R. *Isosurfaces:Geometry, Topology, and Algorithms*. A K Peters/CRC Press, 2013, ch. Chapter 2, pp. 17 – 52.

[11] XIANGSHENG HUANG, XINGHAO CHEM, T. T., AND HUANG, Z. Marching cubes algorithm for fast 3d modeling of human face by incremental data fusion. *Institute of Automation, Chinese Academy of Sciences* (2013).

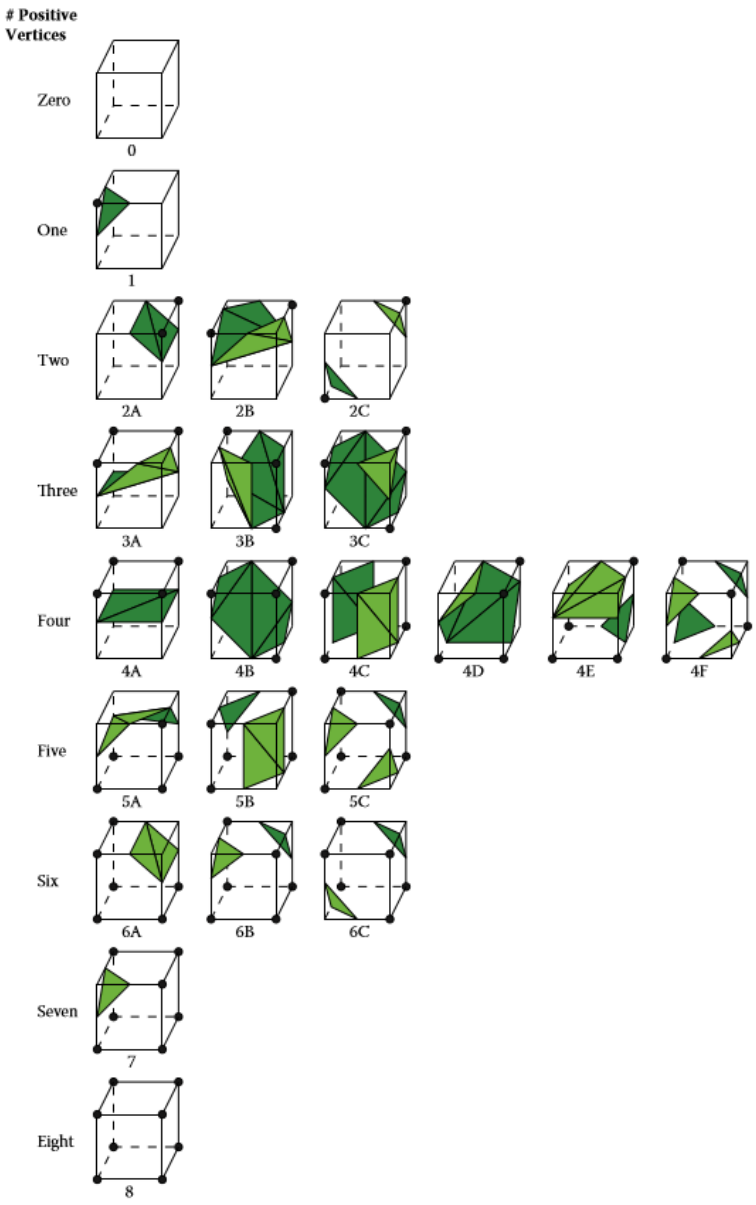
[12] ZIEGLER, G. High-speed marching cubes using histopyramids. *Article in Computer Graphics Forum 26* (2007).

Appendix1



Twenty-two Distinct Vertex Configurations.

Appendix2



Twenty-two Distinct Vertex Configurations.

Appendix3

```
int LookupArray[256] =
{
        0,    265,  515,  778,  1030, 1295, 1541, 1804,
        2060, 2309, 2575, 2822, 3082, 3331, 3593, 3840,
        400,  153,  915,  666,  1430, 1183, 1941, 1692,
        2460, 2197, 2975, 2710, 3482, 3219, 3993, 3728,
        560,  825,  51,   314,  1590, 1855, 1077, 1340,
        2620, 2869, 2111, 2358, 3642, 3891, 3129, 3376,
        928,  681,  419,  170,  1958, 1711, 1445, 1196,
        2988, 2725, 2479, 2214, 4010, 3747, 3497, 3232,
        1120, 1385, 1635, 1898, 102,  367,  613,  876,
        3180, 3429, 3695, 3942, 2154, 2403, 2665, 2912,
        1520, 1273, 2035, 1786, 502,  255,  1013, 764,
        3580, 3317, 4095, 3830, 2554, 2291, 3065, 2800,
        1616, 1881, 1107, 1370, 598,  863,  85,   348,
        3676, 3925, 3167, 3414, 2650, 2899, 2137, 2384,
        1984, 1737, 1475, 1226, 966,  719,  453,  204,
        4044, 3781, 3535, 3270, 3018, 2755, 2505, 2240,
        2240, 2505, 2755, 3018, 3270, 3535, 3781, 4044,
        204,  453,  719,  966,  1226, 1475, 1737, 1984,
        2384, 2137, 2899, 2650, 3414, 3167, 3925, 3676,
        348,  85,   863,  598,  1370, 1107, 1881, 1616,
        2800, 3065, 2291, 2554, 3830, 4095, 3317, 3580,
        764,  1013, 255,  502,  1786, 2035, 1273, 1520,
        2912, 2665, 2403, 2154, 3942, 3695, 3429, 3180,
        876,  613,  367,  102,  1898, 1635, 1385, 1120,
        3232, 3497, 3747, 4010, 2214, 2479, 2725, 2988,
        1196, 1445, 1711, 1958, 170,  419,  681,  928,
        3376, 3129, 3891, 3642, 2358, 2111, 2869, 2620,
        1340, 1077, 1855, 1590, 314,  51,   825,  560,
        3728, 3993, 3219, 3482, 2710, 2975, 2197, 2460,
        1692, 1941, 1183, 1430, 666,  915,  153,  400,
        3840, 3593, 3331, 3082, 2822, 2575, 2309, 2060,
        1804, 1541, 1295, 1030, 778,  515,  265,  0,
};
```

Lookup Array.

Appendix4

```
int LookupMatrix[256][16] = {
        { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 1, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 8, 3, 9, 8, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 8, 3, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 2, 10, 0, 2, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 2, 8, 3, 2, 10, 8, 10, 9, 8, -1, -1, -1, -1, -1, -1, -1 },
        { 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 11, 2, 8, 11, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 9, 0, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 11, 2, 1, 9, 11, 9, 8, 11, -1, -1, -1, -1, -1, -1, -1 },
        { 3, 10, 1, 11, 10, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 10, 1, 0, 8, 10, 8, 11, 10, -1, -1, -1, -1, -1, -1, -1 },
        { 3, 9, 0, 3, 11, 9, 11, 10, 9, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 4, 3, 0, 7, 3, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 1, 9, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 4, 1, 9, 4, 7, 1, 7, 3, 1, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 2, 10, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 3, 4, 7, 3, 0, 4, 1, 2, 10, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 2, 10, 9, 0, 2, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1 },
        { 2, 10, 9, 2, 9, 7, 2, 7, 3, 7, 9, 4, -1, -1, -1, -1 },
        { 8, 4, 7, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 11, 4, 7, 11, 2, 4, 2, 0, 4, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 0, 1, 8, 4, 7, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1 },
        { 4, 7, 11, 9, 4, 11, 9, 11, 2, 9, 2, 1, -1, -1, -1, -1 },
        { 3, 10, 1, 3, 11, 10, 7, 8, 4, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 11, 10, 1, 4, 11, 1, 0, 4, 7, 11, 4, -1, -1, -1, -1 },
        { 4, 7, 8, 9, 0, 11, 9, 11, 10, 11, 0, 3, -1, -1, -1, -1 },
        { 4, 7, 11, 4, 11, 9, 9, 11, 10, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 5, 4, 0, 8, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 5, 4, 1, 5, 0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 8, 5, 4, 8, 3, 5, 3, 1, 5, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 2, 10, 9, 5, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 3, 0, 8, 1, 2, 10, 4, 9, 5, -1, -1, -1, -1, -1, -1, -1 },
        { 5, 2, 10, 5, 4, 2, 4, 0, 2, -1, -1, -1, -1, -1, -1, -1 },
        { 2, 10, 5, 3, 2, 5, 3, 5, 4, 3, 4, 8, -1, -1, -1, -1 },
        { 9, 5, 4, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 11, 2, 0, 8, 11, 4, 9, 5, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 5, 4, 0, 1, 5, 2, 3, 11, -1, -1, -1, -1, -1, -1, -1 },
        { 2, 1, 5, 2, 5, 8, 2, 8, 11, 4, 8, 5, -1, -1, -1, -1 },
        { 10, 3, 11, 10, 1, 3, 9, 5, 4, -1, -1, -1, -1, -1, -1, -1 },
        { 4, 9, 5, 0, 8, 1, 8, 10, 1, 8, 11, 10, -1, -1, -1, -1 },
        { 5, 4, 0, 5, 0, 11, 5, 11, 10, 11, 0, 3, -1, -1, -1, -1 },
        { 5, 4, 8, 5, 8, 10, 10, 8, 11, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 7, 8, 5, 7, 9, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 3, 0, 9, 5, 3, 5, 7, 3, -1, -1, -1, -1, -1, -1, -1 },
        { 0, 7, 8, 0, 1, 7, 1, 5, 7, -1, -1, -1, -1, -1, -1, -1 },
        { 1, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 7, 8, 9, 5, 7, 10, 1, 2, -1, -1, -1, -1, -1, -1, -1 },
        { 10, 1, 2, 9, 5, 0, 5, 3, 0, 5, 7, 3, -1, -1, -1, -1 },
        { 8, 0, 2, 8, 2, 5, 8, 5, 7, 10, 5, 2, -1, -1, -1, -1 },
        { 2, 10, 5, 2, 5, 3, 3, 5, 7, -1, -1, -1, -1, -1, -1, -1 },
        { 7, 9, 5, 7, 8, 9, 3, 11, 2, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 5, 7, 9, 7, 2, 9, 2, 0, 2, 7, 11, -1, -1, -1, -1 },
        { 2, 3, 11, 0, 1, 8, 1, 7, 8, 1, 5, 7, -1, -1, -1, -1 },
        { 11, 2, 1, 11, 1, 7, 7, 1, 5, -1, -1, -1, -1, -1, -1, -1 },
        { 9, 5, 8, 8, 5, 7, 10, 1, 3, 10, 3, 11, -1, -1, -1, -1 },
        { 5, 7, 0, 5, 0, 9, 7, 11, 0, 1, 0, 10, 11, 10, 0, -1 },
```

(a)

```
{ 11, 10, 0, 11, 0, 3, 10, 5, 0, 8, 0, 7, 5, 7, 0, -1 },
{ 11, 10, 5, 7, 11, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 10, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 3, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 0, 1, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 8, 3, 1, 9, 8, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 6, 5, 2, 6, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 6, 5, 1, 2, 6, 3, 0, 8, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 6, 5, 9, 0, 6, 0, 2, 6, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 9, 8, 5, 8, 2, 5, 2, 6, 3, 2, 8, -1, -1, -1, -1 },
{ 2, 3, 11, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 11, 0, 8, 11, 2, 0, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 1, 9, 2, 3, 11, 5, 10, 6, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 10, 6, 1, 9, 2, 9, 11, 2, 9, 8, 11, -1, -1, -1, -1 },
{ 6, 3, 11, 6, 5, 3, 5, 1, 3, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 11, 0, 11, 5, 0, 5, 1, 5, 11, 6, -1, -1, -1, -1 },
{ 3, 11, 6, 0, 3, 6, 0, 6, 5, 0, 5, 9, -1, -1, -1, -1 },
{ 6, 5, 9, 6, 9, 11, 11, 9, 8, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 10, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 3, 0, 4, 7, 3, 6, 5, 10, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 9, 0, 5, 10, 6, 8, 4, 7, -1, -1, -1, -1, -1, -1, -1 },
{ 10, 6, 5, 1, 9, 7, 1, 7, 3, 7, 9, 4, -1, -1, -1, -1 },
{ 6, 1, 2, 6, 5, 1, 4, 7, 8, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 5, 5, 2, 6, 3, 0, 4, 3, 4, 7, -1, -1, -1, -1 },
{ 8, 4, 7, 9, 0, 5, 0, 6, 5, 0, 2, 6, -1, -1, -1, -1 },
{ 7, 3, 9, 7, 9, 4, 3, 2, 9, 5, 9, 6, 2, 6, 9, -1 },
{ 3, 11, 2, 7, 8, 4, 10, 6, 5, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 10, 6, 4, 7, 2, 4, 2, 0, 2, 7, 11, -1, -1, -1, -1 },
{ 0, 1, 9, 4, 7, 8, 2, 3, 11, 5, 10, 6, -1, -1, -1, -1 },
{ 9, 2, 1, 9, 11, 2, 9, 4, 11, 7, 11, 4, 5, 10, 6, -1 },
{ 8, 4, 7, 3, 11, 5, 3, 5, 1, 5, 11, 6, -1, -1, -1, -1 },
{ 5, 1, 11, 5, 11, 6, 1, 0, 11, 7, 11, 4, 0, 4, 11, -1 },
{ 0, 5, 9, 0, 6, 5, 0, 3, 6, 11, 6, 3, 8, 4, 7, -1 },
{ 6, 5, 9, 6, 9, 11, 4, 7, 9, 7, 11, 9, -1, -1, -1, -1 },
{ 10, 4, 9, 6, 4, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 10, 6, 4, 9, 10, 0, 8, 3, -1, -1, -1, -1, -1, -1, -1 },
{ 10, 0, 1, 10, 6, 0, 6, 4, 0, -1, -1, -1, -1, -1, -1, -1 },
{ 8, 3, 1, 8, 1, 6, 8, 6, 4, 6, 1, 10, -1, -1, -1, -1 },
{ 1, 4, 9, 1, 2, 4, 2, 6, 4, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 0, 8, 1, 2, 9, 2, 4, 9, 2, 6, 4, -1, -1, -1, -1 },
{ 0, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 8, 3, 2, 8, 2, 4, 4, 2, 6, -1, -1, -1, -1, -1, -1, -1 },
{ 10, 4, 9, 10, 6, 4, 11, 2, 3, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 2, 2, 8, 11, 4, 9, 10, 4, 10, 6, -1, -1, -1, -1 },
{ 3, 11, 2, 0, 1, 6, 0, 6, 4, 6, 1, 10, -1, -1, -1, -1 },
{ 6, 4, 1, 6, 1, 10, 4, 8, 1, 2, 1, 11, 8, 11, 1, -1 },
{ 9, 6, 4, 9, 3, 6, 9, 1, 3, 11, 6, 3, -1, -1, -1, -1 },
{ 8, 11, 1, 8, 1, 0, 11, 6, 1, 9, 1, 4, 6, 4, 1, -1 },
{ 3, 11, 6, 3, 6, 0, 0, 6, 4, -1, -1, -1, -1, -1, -1, -1 },
{ 6, 4, 8, 11, 6, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 7, 10, 6, 7, 8, 10, 8, 9, 10, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 7, 3, 0, 10, 7, 0, 9, 10, 6, 7, 10, -1, -1, -1, -1 },
{ 10, 6, 7, 1, 10, 7, 1, 7, 8, 1, 8, 0, -1, -1, -1, -1 },
{ 10, 6, 7, 10, 7, 1, 1, 7, 3, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 6, 1, 6, 8, 1, 8, 9, 8, 6, 7, -1, -1, -1, -1 },
{ 2, 6, 9, 2, 9, 1, 6, 7, 9, 0, 9, 3, 7, 3, 9, -1 },
{ 7, 8, 0, 7, 0, 6, 6, 0, 2, -1, -1, -1, -1, -1, -1, -1 },
{ 7, 3, 2, 6, 7, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 2, 3, 11, 10, 6, 8, 10, 8, 9, 8, 6, 7, -1, -1, -1, -1 },
{ 2, 0, 7, 2, 7, 11, 0, 9, 7, 6, 7, 10, 9, 10, 7, -1 },
{ 1, 8, 0, 1, 7, 8, 1, 10, 7, 6, 7, 10, 2, 3, 11, -1 },
{ 11, 2, 1, 11, 1, 7, 10, 6, 1, 6, 7, 1, -1, -1, -1, -1 },
{ 8, 9, 6, 8, 6, 7, 9, 1, 6, 11, 6, 3, 1, 3, 6, -1 },
{ 0, 9, 1, 11, 6, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
```

(b)

Appendix4

```
{ 0, 9, 1, 11, 6, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 7, 8, 0, 7, 0, 6, 3, 11, 0, 11, 6, 0, -1, -1, -1, -1 },
{ 7, 11, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 7, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 0, 8, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 1, 9, 11, 7, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 8, 1, 9, 8, 3, 1, 11, 7, 6, -1, -1, -1, -1, -1, -1 },
{ 10, 1, 2, 6, 11, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 10, 3, 0, 8, 6, 11, 7, -1, -1, -1, -1, -1, -1 },
{ 2, 9, 0, 2, 10, 9, 6, 11, 7, -1, -1, -1, -1, -1, -1 },
{ 6, 11, 7, 2, 10, 3, 10, 8, 3, 10, 9, 8, -1, -1, -1 },
{ 7, 2, 3, 6, 2, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 7, 0, 8, 7, 6, 0, 6, 2, 0, -1, -1, -1, -1, -1, -1 },
{ 2, 7, 6, 2, 3, 7, 0, 1, 9, -1, -1, -1, -1, -1, -1 },
{ 1, 6, 2, 1, 8, 6, 1, 9, 8, 8, 7, 6, -1, -1, -1 },
{ 10, 7, 6, 10, 1, 7, 1, 3, 7, -1, -1, -1, -1, -1, -1 },
{ 10, 7, 6, 1, 7, 10, 1, 8, 7, 1, 0, 8, -1, -1, -1 },
{ 0, 3, 7, 0, 7, 10, 0, 10, 9, 6, 10, 7, -1, -1, -1 },
{ 7, 6, 10, 7, 10, 8, 8, 10, 9, -1, -1, -1, -1, -1, -1 },
{ 6, 8, 4, 11, 8, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 6, 11, 3, 0, 6, 0, 4, 6, -1, -1, -1, -1, -1, -1 },
{ 8, 6, 11, 8, 4, 6, 9, 0, 1, -1, -1, -1, -1, -1, -1 },
{ 9, 4, 6, 9, 6, 3, 9, 3, 1, 11, 3, 6, -1, -1, -1 },
{ 6, 8, 4, 6, 11, 8, 2, 10, 1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 10, 3, 0, 11, 0, 6, 11, 0, 4, 6, -1, -1, -1 },
{ 4, 11, 8, 4, 6, 11, 0, 2, 9, 2, 10, 9, -1, -1, -1 },
{ 10, 9, 3, 10, 3, 2, 9, 4, 3, 11, 3, 6, 4, 6, 3, -1 },
{ 8, 2, 3, 8, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1 },
{ 0, 4, 2, 4, 6, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 9, 0, 2, 3, 4, 2, 4, 6, 4, 3, 8, -1, -1, -1 },
{ 1, 9, 4, 1, 4, 2, 2, 4, 6, -1, -1, -1, -1, -1, -1 },
{ 8, 1, 3, 8, 6, 1, 8, 4, 6, 6, 10, 1, -1, -1, -1 },
{ 10, 1, 0, 10, 0, 6, 6, 0, 4, -1, -1, -1, -1, -1, -1 },
{ 4, 6, 3, 4, 3, 8, 6, 10, 3, 0, 3, 9, 10, 9, 3, -1 },
{ 10, 9, 4, 6, 10, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 9, 5, 7, 6, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 3, 4, 9, 5, 11, 7, 6, -1, -1, -1, -1, -1, -1 },
{ 5, 0, 1, 5, 4, 0, 7, 6, 11, -1, -1, -1, -1, -1, -1 },
{ 11, 7, 6, 8, 3, 4, 3, 5, 4, 3, 1, 5, -1, -1, -1 },
{ 9, 5, 4, 10, 1, 2, 7, 6, 11, -1, -1, -1, -1, -1, -1 },
{ 6, 11, 7, 1, 2, 10, 0, 8, 3, 4, 9, 5, -1, -1, -1 },
{ 7, 6, 11, 5, 4, 10, 4, 2, 10, 4, 0, 2, -1, -1, -1 },
{ 3, 4, 8, 3, 5, 4, 3, 2, 5, 10, 5, 2, 11, 7, 6, -1 },
{ 7, 2, 3, 7, 6, 2, 5, 4, 9, -1, -1, -1, -1, -1, -1 },
{ 9, 5, 4, 0, 8, 6, 0, 6, 2, 6, 8, 7, -1, -1, -1 },
{ 3, 6, 2, 3, 7, 6, 1, 5, 0, 5, 4, 0, -1, -1, -1 },
{ 6, 2, 8, 6, 8, 7, 2, 1, 8, 4, 8, 5, 1, 5, 8, -1 },
{ 9, 5, 4, 10, 1, 6, 1, 7, 6, 1, 3, 7, -1, -1, -1 },
{ 1, 6, 10, 1, 7, 6, 1, 0, 7, 8, 7, 0, 9, 5, 4, -1 },
{ 4, 0, 10, 4, 10, 5, 0, 3, 10, 6, 10, 7, 3, 7, 10, -1 },
{ 7, 6, 10, 7, 10, 8, 5, 4, 10, 4, 8, 10, -1, -1, -1, -1 },
{ 6, 9, 5, 6, 11, 9, 11, 8, 9, -1, -1, -1, -1, -1, -1 },
{ 3, 6, 11, 0, 6, 3, 0, 5, 6, 0, 9, 5, -1, -1, -1, -1 },
{ 0, 11, 8, 0, 5, 11, 0, 1, 5, 5, 6, 11, -1, -1, -1, -1 },
{ 6, 11, 3, 6, 3, 5, 5, 3, 1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 10, 9, 5, 11, 9, 11, 8, 11, 5, 6, -1, -1, -1, -1 },
{ 0, 11, 3, 0, 6, 11, 0, 9, 6, 5, 6, 9, 1, 2, 10, -1 },
{ 11, 8, 5, 11, 5, 6, 8, 0, 5, 10, 5, 2, 0, 2, 5, -1 },
{ 6, 11, 3, 6, 3, 5, 2, 10, 3, 10, 5, 3, -1, -1, -1, -1 },
{ 5, 8, 9, 5, 2, 8, 5, 6, 2, 3, 8, 2, -1, -1, -1, -1 },
{ 9, 5, 6, 9, 6, 0, 0, 6, 2, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 5, 8, 1, 8, 0, 5, 6, 8, 3, 8, 2, 6, 2, 8, -1 },
{ 1, 5, 6, 2, 1, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 3, 6, 1, 6, 10, 3, 8, 6, 5, 6, 9, 8, 9, 6, -1 },
{ 10, 1, 0, 10, 0, 6, 9, 5, 0, 5, 6, 0, -1, -1, -1, -1 },
{ 0, 3, 8, 5, 6, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
```

(a)

```
{ 10, 5, 6, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 11, 5, 10, 7, 5, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 11, 5, 10, 11, 7, 5, 8, 3, 0, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 11, 7, 5, 10, 11, 1, 9, 0, -1, -1, -1, -1, -1, -1, -1 },
{ 10, 7, 5, 10, 11, 7, 9, 8, 1, 8, 3, 1, -1, -1, -1, -1 },
{ 11, 1, 2, 11, 7, 1, 7, 5, 1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 3, 1, 2, 7, 1, 7, 5, 7, 2, 11, -1, -1, -1, -1 },
{ 9, 7, 5, 9, 2, 7, 9, 0, 2, 2, 11, 7, -1, -1, -1, -1 },
{ 7, 5, 2, 7, 2, 11, 5, 9, 2, 3, 2, 8, 9, 8, 2, -1 },
{ 2, 5, 10, 2, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1, -1 },
{ 8, 2, 0, 8, 5, 2, 8, 7, 5, 10, 2, 5, -1, -1, -1, -1 },
{ 9, 0, 1, 5, 10, 3, 5, 3, 7, 3, 10, 2, -1, -1, -1, -1 },
{ 9, 8, 2, 9, 2, 1, 8, 7, 2, 10, 2, 5, 7, 5, 2, -1 },
{ 1, 3, 5, 3, 7, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 7, 0, 7, 1, 1, 7, 5, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 0, 3, 9, 3, 5, 5, 3, 7, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 8, 7, 5, 9, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 8, 4, 5, 10, 8, 10, 11, 8, -1, -1, -1, -1, -1, -1, -1 },
{ 5, 0, 4, 5, 11, 0, 5, 10, 11, 11, 3, 0, -1, -1, -1, -1 },
{ 0, 1, 9, 8, 4, 10, 8, 10, 11, 10, 4, 5, -1, -1, -1, -1 },
{ 10, 11, 4, 10, 4, 5, 11, 3, 4, 9, 4, 1, 3, 1, 4, -1 },
{ 2, 5, 1, 2, 8, 5, 2, 11, 8, 4, 5, 8, -1, -1, -1, -1 },
{ 0, 4, 11, 0, 11, 3, 4, 5, 11, 2, 11, 1, 5, 1, 11, -1 },
{ 0, 2, 5, 0, 5, 9, 2, 11, 5, 4, 5, 8, 11, 8, 5, -1 },
{ 9, 4, 5, 2, 11, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 2, 5, 10, 3, 5, 2, 3, 4, 5, 3, 8, 4, -1, -1, -1, -1 },
{ 5, 10, 2, 5, 2, 4, 4, 2, 0, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 10, 2, 3, 5, 10, 3, 8, 5, 4, 5, 8, 0, 1, 9, -1 },
{ 5, 10, 2, 5, 2, 4, 1, 9, 2, 9, 4, 2, -1, -1, -1, -1 },
{ 8, 4, 5, 8, 5, 3, 3, 5, 1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 4, 5, 1, 0, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 8, 4, 5, 8, 5, 3, 9, 0, 5, 0, 3, 5, -1, -1, -1, -1 },
{ 9, 4, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 11, 7, 4, 9, 11, 9, 10, 11, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 8, 3, 4, 9, 7, 9, 11, 7, 9, 10, 11, -1, -1, -1, -1 },
{ 1, 10, 11, 1, 11, 4, 1, 4, 0, 7, 4, 11, -1, -1, -1, -1 },
{ 3, 1, 4, 3, 4, 8, 1, 10, 4, 7, 4, 11, 10, 11, 4, -1 },
{ 4, 11, 7, 9, 11, 4, 9, 2, 11, 9, 1, 2, -1, -1, -1, -1 },
{ 9, 7, 4, 9, 11, 7, 9, 1, 11, 2, 11, 1, 0, 8, 3, -1 },
{ 11, 7, 4, 11, 4, 2, 2, 4, 0, -1, -1, -1, -1, -1, -1, -1 },
{ 11, 7, 4, 11, 4, 2, 8, 3, 4, 3, 2, 4, -1, -1, -1, -1 },
{ 2, 9, 10, 2, 7, 9, 2, 3, 7, 7, 4, 9, -1, -1, -1, -1 },
{ 9, 10, 7, 9, 7, 4, 10, 2, 7, 8, 7, 0, 2, 0, 7, -1 },
{ 3, 7, 10, 3, 10, 2, 7, 4, 10, 1, 10, 0, 4, 0, 10, -1 },
{ 1, 10, 2, 8, 7, 4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 9, 1, 4, 1, 7, 7, 1, 3, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 9, 1, 4, 1, 7, 0, 8, 1, 8, 7, 1, -1, -1, -1, -1 },
{ 4, 0, 3, 7, 4, 3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 4, 8, 7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 10, 8, 10, 11, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 0, 9, 3, 9, 11, 11, 9, 10, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 1, 10, 0, 10, 8, 8, 10, 11, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 1, 10, 11, 3, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 2, 11, 1, 11, 9, 9, 11, 8, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 0, 9, 3, 9, 11, 1, 2, 9, 2, 11, 9, -1, -1, -1, -1 },
{ 0, 2, 11, 8, 0, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 3, 2, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 2, 3, 8, 2, 8, 10, 10, 8, 9, -1, -1, -1, -1, -1, -1, -1 },
{ 9, 10, 2, 0, 9, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 2, 3, 8, 2, 8, 10, 0, 1, 8, 1, 10, 8, -1, -1, -1, -1 },
{ 1, 10, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 1, 3, 8, 9, 1, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 9, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ 0, 3, 8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }
};
```

(b)

Lookup Matrix.