Computer Graphics-USTC-2019

This is a a report about the projects the author complete when follows Ligang Liu's CG courses online.
One can view the course on the website :USTC-CG-2019-Spring(Click)
The core code link : repository of the source code(Click)

Contributors :
Author
Shihan Cheng PB19000216

Instructor :
Ligang Liu Details

Sommaire

Liste des figures

Liste des tableaux

1. OOP

1.1. Minidraw

This is a warming-up exercise which enhance the realization of **OOP** ,the Object-oriented Programming.

1.1.1. Requests

1. Implement a drawing applet "MiniDraw" that requires drawing drawing elements (Line), ellipse (Ellipse), rectangle (Rectangle), polygon (Polygon) (element), freehand (Freehand), etc.
2. Each element is encapsulated with a class (object), such as CLine, CEllipse, CRect, CPolygon, and CFreehand.
3. Various elements inherit from a parent class, such as CFigure.
4. Learning class inheritance and polymorphism.

1.1.2. Introduction

Minidraw

MiniDraw is a simple Qt GUI application which allows users to draw figures including lines, triangles, ellipses, rectangles, rounded rectangles, polygons, arrows and traces. Users can also select figures and move them, rotate them, resize them or delete them. Colors and stroke width are also allowed to change. User can also choose to draw the same type of figures consecutively or choose not to do so.

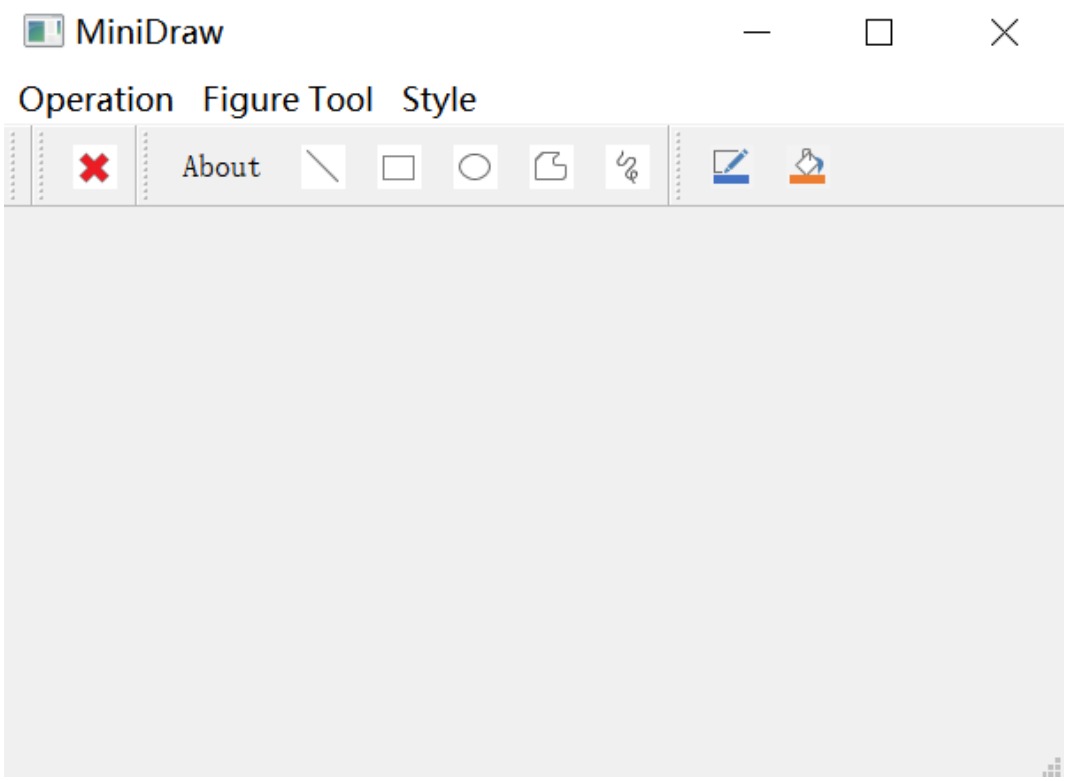

FIGURE 1 – HW1-MInidraw-UI

1.1.3. Demo

To draw normal figures (figures except polygons), choose the corresponding figure from "Figure" menu or "Figure" toolbar. Press the left button of the mouse to start drawing. Drag the mouse to draw. Release the

mouse to complete drawing. The figure you have just drawn will be painted and selected.
To draw polygons, choose "Polygon" from "Figure" menu or "Figure" toolbar. Press the left button of the mouse to start drawing. Drag the mouse to draw the first edge. Release the mouse to set the end point of the edge. Then drag the mouse and click for the next vertex. The rest vertices can be done in the same manner. To draw the last vertex and complete drawing, press the right button of the mouse. To select a figure, choose the "Select" option from "Operation" menu or "Operation" toolbar. Move the mouse to a figure and click to select it. When a figure is selected, its frame will be drawn, and then it can be repainted or deleted. [8]

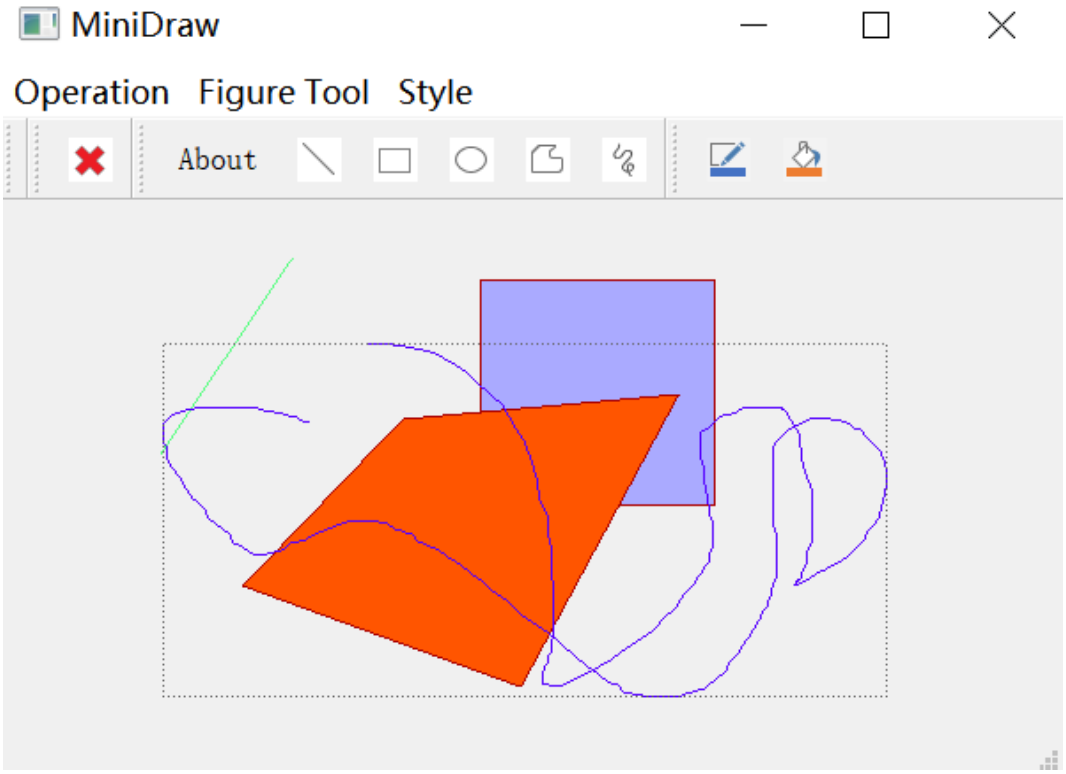

FIGURE 2 – HW1-Minidraw-demo

1.1.4. Classdiagram

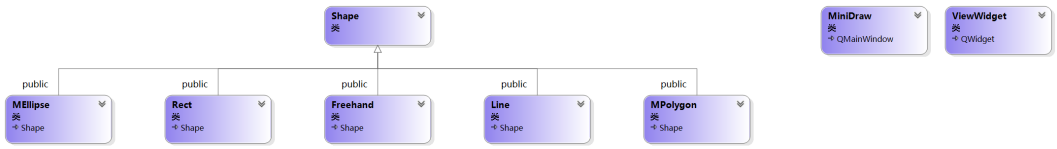

FIGURE 3 – HW1-MInidraw-classdiagram

2. Image

2.1. ImageWarping

2.1.1. Requests

Image Warping

Image warping is a transformation that is applied to the domain of an image, which modifies the geometrical properties of the image itself. Ideally, the intensity of the warped image is the same as the intensity of the original image at corresponding points. In this project, one need to complete two famous algorithms

1. IDW [8] : Inverse distance-weighted interpolation methods
2. RBF [1] :Radial basis functions interpolation method

Also, you need too fix the white line problem of your warping class.

2.1.2. Description

The term "Image Warping" describes the methods for deforming images to arbitrary shapes. By using the control points, the problem of image warping is essentially the problem of scattered data interpolation.

— Input : n pairs $(\mathbf{p}_i, \mathbf{q_i})$ of control points, $\mathbf{p}_i, \mathbf{q}_i \in \mathbb{R}^2, i = 1, \ldots, n$.

— Output : An at-least-continuous function $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$ with $\mathbf{f}(\mathbf{p}_i) = \mathbf{q}_i, i = 1, \ldots, n$

2.1.3. IDW

Inverse distance-weighted interpolation methods were originally proposed by Shepard and improved by a number of other authors, notably Franke and Nielson.

For each control point pi, we use local approximation $\mathbf{f}_i(\mathbf{p}) : \mathbb{R}^2 \to \mathbb{R}^2$ with $f_i(\mathbf{p}_i) = \mathbf{q}_i$ and set the interpolation function a weighted average of these local approximations :

$$\mathbf{f}(\mathbf{x}) = \sum_{i=1}^{n} w_i(\mathbf{x}) \mathbf{f}_i(\mathbf{x}) \tag{1}$$

where $w_i(p)$ is the weight function, which must satisfy the condition :

$$0 \leq w_i(\boldsymbol{x}) \leq 1 \quad \sum_{i=1}^{n} w_i(\mathbf{x}) = 1 \tag{2}$$

These conditions guarantee the property of interpolation. Shepard proposed the following simple weight function :

$$w_i(\mathbf{x}) = \frac{\sigma_i(\mathbf{x})}{\sum_{j=1}^{n} \sigma_j(\mathbf{p})} \quad with \quad \sigma_i(\mathbf{x}) = \frac{1}{\|\mathbf{x} - \mathbf{x}_i\|^{\mu}} \tag{3}$$

where $d(p, pi)$is the distance between p and p_i. usually, we set $\mu = 2$, more simpiliy, $\mathbf{D}_i = \mathbf{0}$, and $\mathbf{f}(\mathbf{x}) = \sum_{i=1}^{n} w_i(\mathbf{x}) \mathbf{q}_i$ .Define the energy as follow :

$$\begin{aligned} E_i(\mathbf{D}_i) &= \sum_{j=1, j\neq i}^{n} w_{ij} \left\| \mathbf{q}_i + \begin{pmatrix} d_{i,11} & d_{i,12} \\ d_{i,21} & d_{i,22} \end{pmatrix} (\mathbf{p}_j - \mathbf{p}_i) - \mathbf{q}_j \right\|^2 \\ &= \sum_{j=1, j\neq i}^{n} w_{ij} ((d_{i,11}(p_{j,1} - p_{i,1}) + d_{i,12}(p_{j,2} - p_{i,2}) + q_{i,1} - q_{j,1})^2 + \\ &\quad (d_{i,21}(p_{j,1} - p_{i,1}) + d_{i,22}(p_{j,2} - p_{i,2}) + q_{i,2} - q_{j,2})^2) \end{aligned} \tag{4}$$

Minimize the energy, we derive that $\mathbf{D}_i = \begin{pmatrix} d_{i,11} & d_{i,12} \\ d_{i,21} & d_{i,22} \end{pmatrix}$.

RBF Transformations based on radial basis functions have proven to be a powerful tool in image warping. It has the expression below :

$$f(p) = A(p) + R(p) \tag{5}$$

where A(p) = M p+b is a D affine transformation (M is a 2 x 2 real matrix), and R(p) is a radial transformation defined by $R(p) = (R_x(p), R_y(p))$. $R_x(p), R_y(p)$ are both radial functions of the form :

$$R(p) = \sum_{i=1}^{n} a_i g_i(||p - p_i||) \tag{6}$$

$g : \boldsymbol{R}^+ \rightarrow \boldsymbol{R}$ is a univariate function, termed the radial basis function. So, f(p) is determined by 2(n+ 3) coefficients : 6 for the affine part, and 2n for the radial components. We define an imagewarping transformation based on the mapping of n anchor points :

$$f(pi) = qi \quad i = 1, 2, \ldots n \tag{7}$$

These interpolation conditions translate into 2n linear equations in the coefficients of the f(p), thus leaving 6 degrees of freedom (3 for each dimension).
By adding the additional equations, one can mark two sets of anchor points. One set (the affine set, red line in project) determines the affine part of the mapping, while the other set (the radial set, green line in project) controls the generation of changes in the facial expression. Note that these sets may intersect, and may even be identical.
f anchor points in the affine set, as follows. If no points are specified, the affine component is the identity mapping ; if one point is specified - translation ; two points - translation and scaling ; three points - general affine transformation ; more than three - general affine transformation determined by a least-square approximation procedure. Then, $R(pi) = qiA(pi) \quad i = 1 \ldots n$ can be solved by choicing the radial basis functions like

$$g(d) = (d^2 + r^2)^{\frac{1}{2}} \tag{8}$$

Following a suggestion by Eck, we used individual values ri for each point pi, computed from the distance to the nearest neighbor :

$$r_i = min_{i \neq j} d(p_j, p_i) \tag{9}$$

2.1.4. Fillhole

For point a in the white seam, select the 8 pixels around point a and average the color of the normal pixels in the 8 pixels to get the color of point a. However, I should also point out that this method is flawed, especially for these image with a wide white beamer.

2.1.5. Demo

The follow four picture is the demo of the imagewarping of a cessboard image using IDW and RBF method. The left image simply apply IDW or RBF method without fillhole, while the right image apply fillhole on the image after IDW or RBF. It ;s obvious that the fillhole method derive a better result. One can see from the result of the imagewarping results that RBF get a more satisfied result compare to the IDW, which bring less white beams, hence the fillhole leads to a more unflawed result.

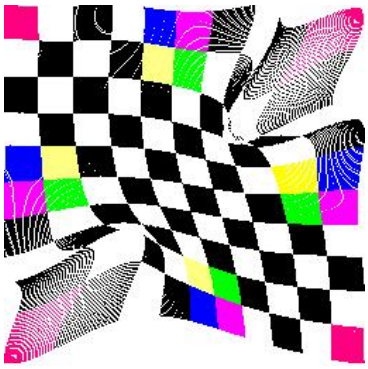

FIGURE 4 – HW2-IDW-chessboard-nofillhole

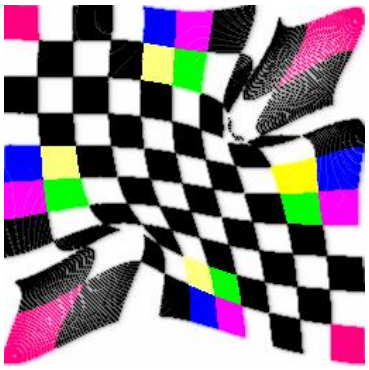

FIGURE 5 – HW2-IDW-chessboard-fillhole

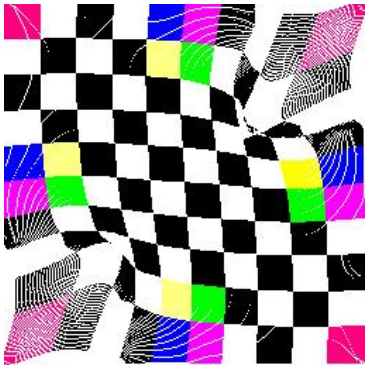

FIGURE 6 – HW2-IDW-chessboard-nofillhole

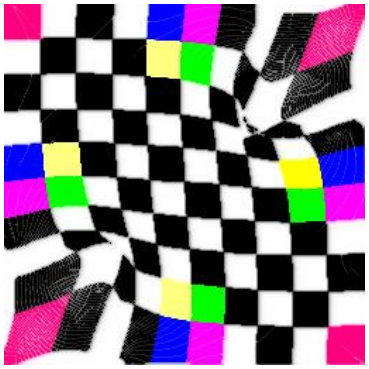

FIGURE 7 – HW2-IDW-chessboard-fillhole

By adding more control points, we can derive some interesting results using imagewarping program. Here is a demo of the image warping of the image of the **Mona Lisa Smile**. I lift up the lip of both sides, and a funny smile appears.

FIGURE 8 – HW2-Smile-demo

2.1.6. Classdiagram

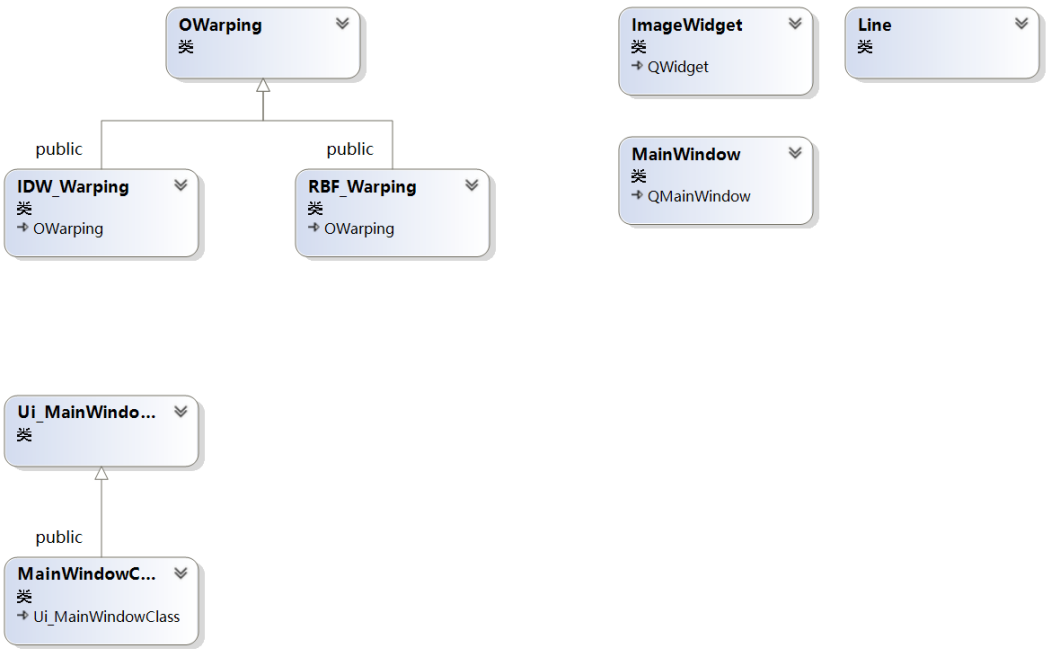

FIGURE 9 – HW2-ImageWarping-classdiagram

2.2. Possion-Editing

2.2.1. Request

Image-Editing

A common task of image editing is to port one image to another.Direct replication is generally difficult to fuse into the graph.How can you seamlessly integrate the source image, such as a target image ?This problem can be abstracted as : the color value of target image is a smooth function on the 2-dimensional region (closure),$f* : D \rightarrow R, D \subseteq R^2$, solve the 2-dimensional function$f : S \rightarrow R \quad S \subset D$, allowing it to maintain some smooth properties with the boundary, while having some phases with source $g\omega \rightarrow R$ similar characteristics.This is equivalent to a bootstrap interpolation problem.

1. Apply the "Possion Image Editing" algorithm in your own program, which is published in Sigraph 2003. [7], also you need to use the scan-to-line method when the boundary is polygon [1].
2. Explore deeper into muti-implemention of Possion image editing. For example, try to construct your program a real-time editing one.

2.2.2. Alogorithm

Denote the boundary of $\partial\Omega$ as, $\Omega = S \backslash \partial\Omega$, the boundary equation can be written as

$$f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{10}$$

We want the gradient to be as similar as the gradient field $\mathbf{v}$, that is, to solve a variational problem :

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 d\Omega \quad \text{with} \quad f|_{\partial\omega} = f^*|_{\partial\Omega} \tag{11}$$

The solution to this variational problem is equivalent to that of the following equation.

1. https ://www.cnblogs.com/mzyan/p/9704008.html

$$\Delta f = div\mathbf{v} \text{ over } \Omega \quad \text{with} \quad f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{12}$$

This is the Poisson equation for the Dirichlet boundary condition, with the unique solution.In particular, say, if the gradient field is the gradient field of the source image,$\mathbf{v} = \nabla g$,

$$\Delta f = \Delta g \; over \; \Omega, \; with \; f|_{\partial\omega} = f^*|_{\partial\Omega} \tag{13}$$

Since the image are composed of discrete pixels, we can transfer the problem to the discrete points, ie :

$$\min_{f|\Omega} \sum_{(p,q)|\cap\Omega\neq\emptyset} (f_p - f_q - v_{pq})^2, \text{ with} f_p = f_p*, \text{for all } p \in \partial\Omega \tag{14}$$

which solution is :

$$\forall p \in \Omega, \; |N_p| f_p - \sum_{q_p \cap \Omega} f_q = \sum_{q_p \cap \partial\omega} f_q^* + \sum_{q \in N_p} v_{pq} \tag{15}$$

where N_p refers to the neighbors of p. Hence we derive a sparse linear function $\mathbf{AX} = \mathbf{b}$, The diagonal element of the coefficient matrix is N_p, other elments are -1 or 0; what ;s more,if p is the neighbor of q, then q is the neighbor of p as well. So, matrix **A** is a Sparse positive-definite symmetric square matrix. The equation has wide applicability and can be solved in graphics.For the issue of image fusion, a single pixel has 4 neighbors around it. It is therefore known that at most only four elements in our matrix are not zero. It is generally very large, so it can not be solved with the algorithm for solving the dense matrix (system of equations), because generally, its complexity is $\backslash^{\ni}$, Some numerical methods of sparse matrices should be used, being able to solve the problem better.
There are also some problems in the application of image fusion, some features are not significant or important in the source graph, we want to transplant more significant features to the target, while those not significant features are not required, which can be achieved through a mixed gradient (Mixed Gradient).The gradient values of the source image and the gradient selection of the target figure are larger, which can reflect the more significant one between the two images.

$$v_{pq} = \begin{cases} f_p^* - f_q^*, & \text{if} |f_p^* - f_q^*| > |g_p - g_q| \\ g_p - g_q, & \text{otherwise} \end{cases} \tag{16}$$

Solving the system of linear equations = is still at a relatively large cost.In some cases of user operation, the source image needs to be dragged to find the appropriate location.At this time, the points where the source image is to be fused do not change, and the coefficient matrix is constant, so we can adopt the matrix predecomposition method, solve it more quickly, and even calculate the pixel color value immediately. We apply Cholesky decomposition here.

2.2.3. Demo

In the demo, we include two pictures, the girl in the water11 and the bear in the water10 to be the source images, while there is also a new background, the possion-editing help us to blend the source images into the new background12. The following two set of pictures shows the difference between the plain possion-editing and the mixed possion editing. We can observe that mixed possion editing preserve more details about the source images.
The real-time possion editing : HW3-real-time-Possion(Click)

FIGURE 10 – HW3-source-BearInWater

FIGURE 11 – HW3-source-GirlInWater

FIGURE 12 – HW3-NewBackGround

FIGURE 13 – HW3-demo-Possion

FIGURE 14 – HW3-demo-mixed-Possion

2.2.4. ClassDiagram

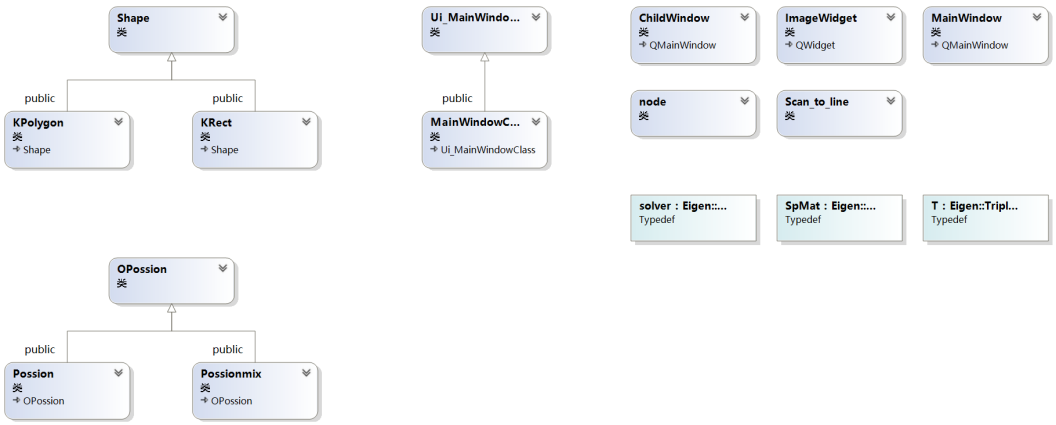

FIGURE 15 – HW3-ClassDiagram

3. Geometry

3.1. Minimal Surfaces and Mesh Parameterization

3.1.1. Requests

Introduction

The representation of surfaces and shapes in the computer is approximated by some discrete sampling points and by grids composed of edges, faces and bodies.A data structure of a storage grid is a half-sided data structure Half Edge Structure and can represent a two-dimensional manifold surface.Half the data structure can easily access the neighbors, neighbors and faces on the point, and also separates the coordinates of the points and the connection between the points, making it more convenient to handle the deformation of the grid structure.Some geometric problems need to be transformed into corresponding discrete methods to treat the grid.This job contains two items Two minimal surfaces, surfaces parameterization of two contents.Minimal surfaces are surfaces with average curvature $\frac{1}{2}(\kappa_1 + \kappa_2)$ everywhere 0, the smallest area that satisfy some constraints (e. g., boundary constraints).Parameterization refers to giving parameters to points on a two-dimensional manifold surface, applied to the material map.

1. Preliminary understanding of the *.obj data (*. obj,*. mtl)
2. Implement the algorithm of the paper [2]

3.1.2. algorithm

Minisurface defined in math is a surface with curvature 0 every where. In a computer, the problem needs to correspond to a grid to solve : consider γ to be the closed curve surrounded on a point v_i on the surface.

$$\lim_{len(\gamma)\to 0} \frac{1}{len(\gamma)} \int_{v\in\gamma} (v_i - v)\mathrm{d}s = H(v_i)\boldsymbol{n}_i \tag{17}$$

Hence we derive that the Differential coordinates satisfy

$$\delta_i = v_i - \frac{1}{d_i} \sum_{v\in N(i)} v = \frac{1}{d_i} \sum_{v\in N(i)} (v_i - v) = 0 \tag{18}$$

Thus for each non-constrained point an equation is obtained with its neighbors.The equation of the constraint point is set at = 0, equivalent to the constraint point in the coefficient matrix corresponds to a diagonal element of 1. To do so, the equation is positive definite asymmetric and can only perform coefficient matrix

decomposition and solve the LU method.
Maps are important concepts in 3 D rendering.Map can not only be diffuse maps, but also express more surface features, such as normal maps, smoothness maps.The map depends on the parameterization of the surface or mesh.Parameterization of a surface (grid) can be parameterized by mapping the surface (grid) to a plane, and then in two-dimensional coordinates on the plane.This plane can be solved by Laplacian coordinates, with 0 Laplacian zero for each inner point. Similar to minimization surfaces seeking a constrained boundary.

$$\delta_i = v_i - \sum_{v_j \in N(v_i)} \omega_j v_j = 0, \text{ with } \sum_{j=1}^{d_i} \omega_j = 1 \tag{19}$$

This method requires a surface with at least 1 boundary.The algorithm works the best for only one boundary, and the surface is close to the level.When the surface is uneven, if the middle has a large bulge compared to the boundary, presents the bottle port structure, the mapped to the middle area on the plane grid will be very dense, so that the parameterization of the internal area will be obviously small step length.The effect presented on the map is that the middle area map is overstretched method.For multi-boundary surfaces, the algorithm can be done, but the mapping effect will become odd.For multi-boundary, high-loss surfaces, some surfaces should be divided and cut The algorithm, divided into multiple surfaces, then the map effect will be better.A good parameterization requires that 1. The surface does not flip on the mapping to the plane.According to the theorem, any centroid coordinate map of the convex boundary guarantees that each point is inside the polygon composed of its neighbors, that means that the grid does not flip, so this condition can be satisfied only by mapping the boundary to the convex polygon ; 2. Can maintain the original geometric relationship to some extent.This method solves two factors depending on the topology of the grid and the selection of the centroid coordinate weight.

1. **Uniform Weight** It is easier to take average weights $\omega_j = \frac{1}{d_i}$

$$\delta_i = v_i - \frac{1}{d_i} \sum_{v_j(v_i)} v_j = 0 \tag{20}$$

   This method only determines the effect of parameterization by the topology of the original grid, and does not exploit the geometry of the grid, remaining poorly on the grid shape after mapping to the plane, especially for inhomogeneous grids.As can be seen in the later test results, the mapping effect of the average weight method is visually poor.

2. **Cotangent Weight** The weight in this method is set to be $\omega_j = cot\alpha + cot\beta$

$$\delta_i = v_i - \frac{1}{\sum w_j} \sum_{v_j \in N(v_i)} \omega_j v_j = 0, w_j = cot\alpha_j + cot\beta_j \tag{21}$$

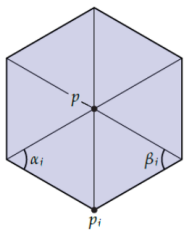

Figure 16 – HW4-tangent-image

3.1.3. Demo

This is a description of the procedure. We can load the required obj files via Hierarchy-root-mesh-load-sobj. In addition, you have two ways to observe the obj file you are importing. By changing the Setting-Viewer-Rastertype, you can observe the results of the triangular mesh structure19 and texture mapping17 of the 3D object. The Transform bar on the right is used to change the spatial position of the loaded object, and the Material bar can implement texture mapping of 3D objects. The Geometry bar is a few of my own defined operations that encapsulate the minimal and parametric surface operations in HW4, and of course, since HW5-HW6 is also used in this program, it also encapsulates some other interesting operations (mentioned in subsequent reports).
The naming conventions for the program function interfaces encapsulated in HW-4 are as follows : Square or Circle that appears in the naming indicates mapping the 3D mesh to a square or circle ; uniform or cotangent indicates whether it uses the averaging weighting method or the weighting method of trigonometric cotangent ; whether display is used as a suffix to indicate whether to perform a texture mapping operation this time, and if a texture mapping operation is required, You need to first import the textures you need to map through the Material-Albedo Texture column.
For more information, you can view the demo video here : HW4-video-demo(Click)

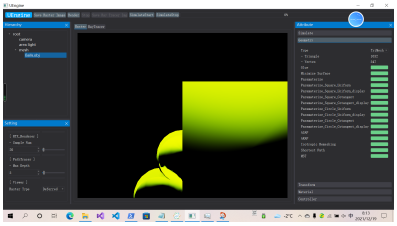

FIGURE 17 – HW4-UI-Deffered

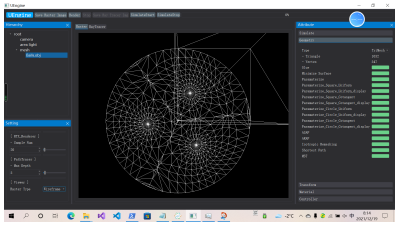

FIGURE 18 – HW4-UI-wireframe

Demo shows the difference when implenment the minisurface operation on a ball.

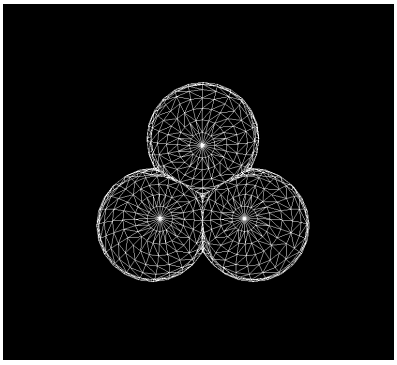

FIGURE 19 – HW4-initial-ball

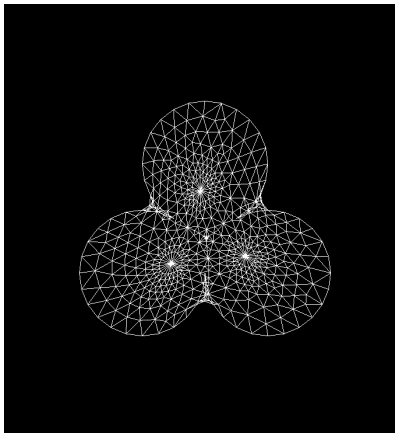

FIGURE 20 – HW4-ball-minisurface

To compare the difference betweeen two weighted method, I tested several objects, and the results are listed as follows. As you can see, the trigonometric cotangent method2224 beat the average weighted method2123.

From left to right, according to this is an untexed map, a proto-triangular mesh structure, an average-weighted square map and its corresponding texture map. The 2D texture is the colored checkerboard map used in ImageWarping.

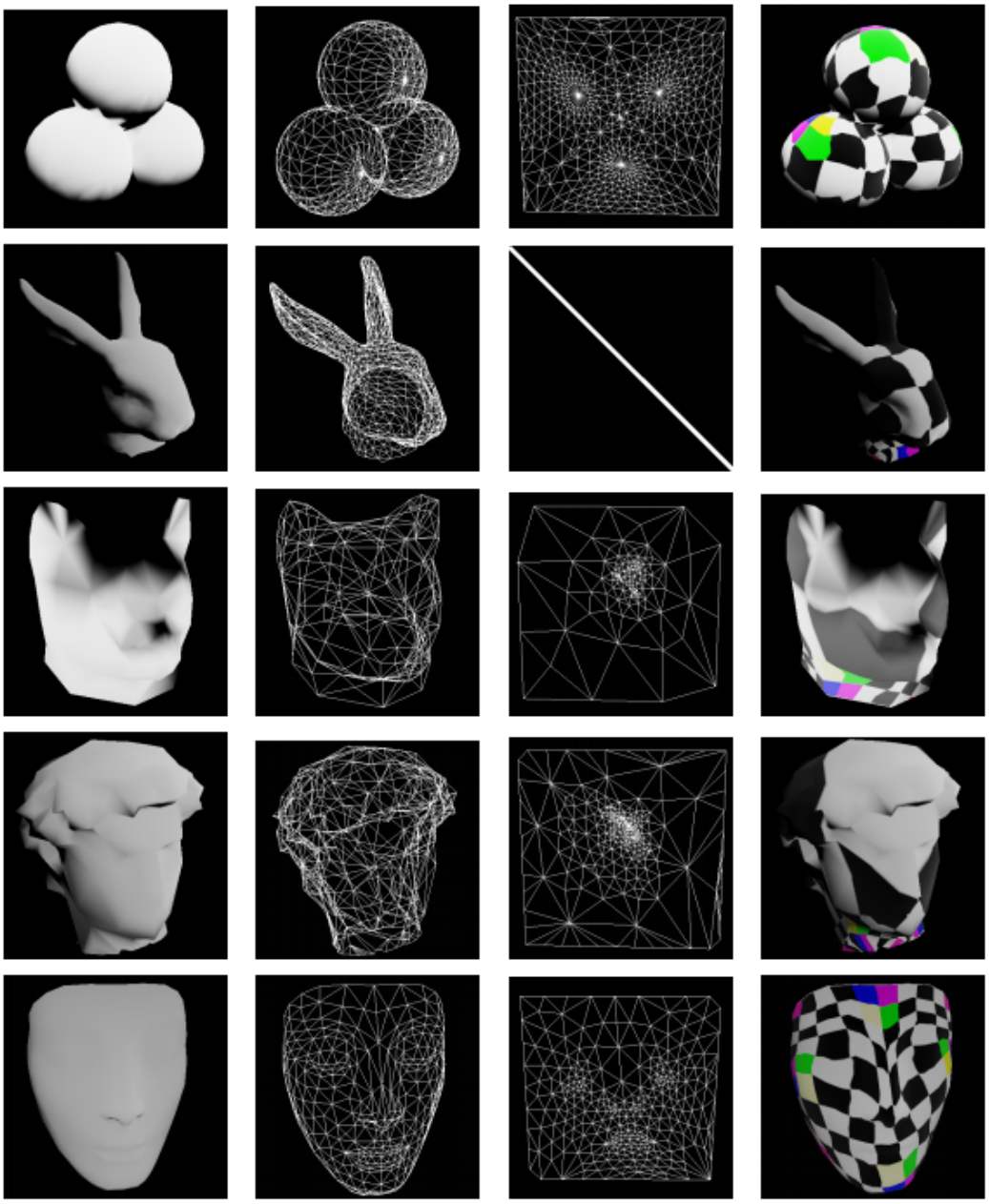

FIGURE 21 – HW4-uniform-gptm-square

From left to right, according to this is the untexed map, the original triangular mesh structure, the cotangent-weighted square map and its corresponding texture map. The 2D texture is the colored checkerboard map used in ImageWarping.

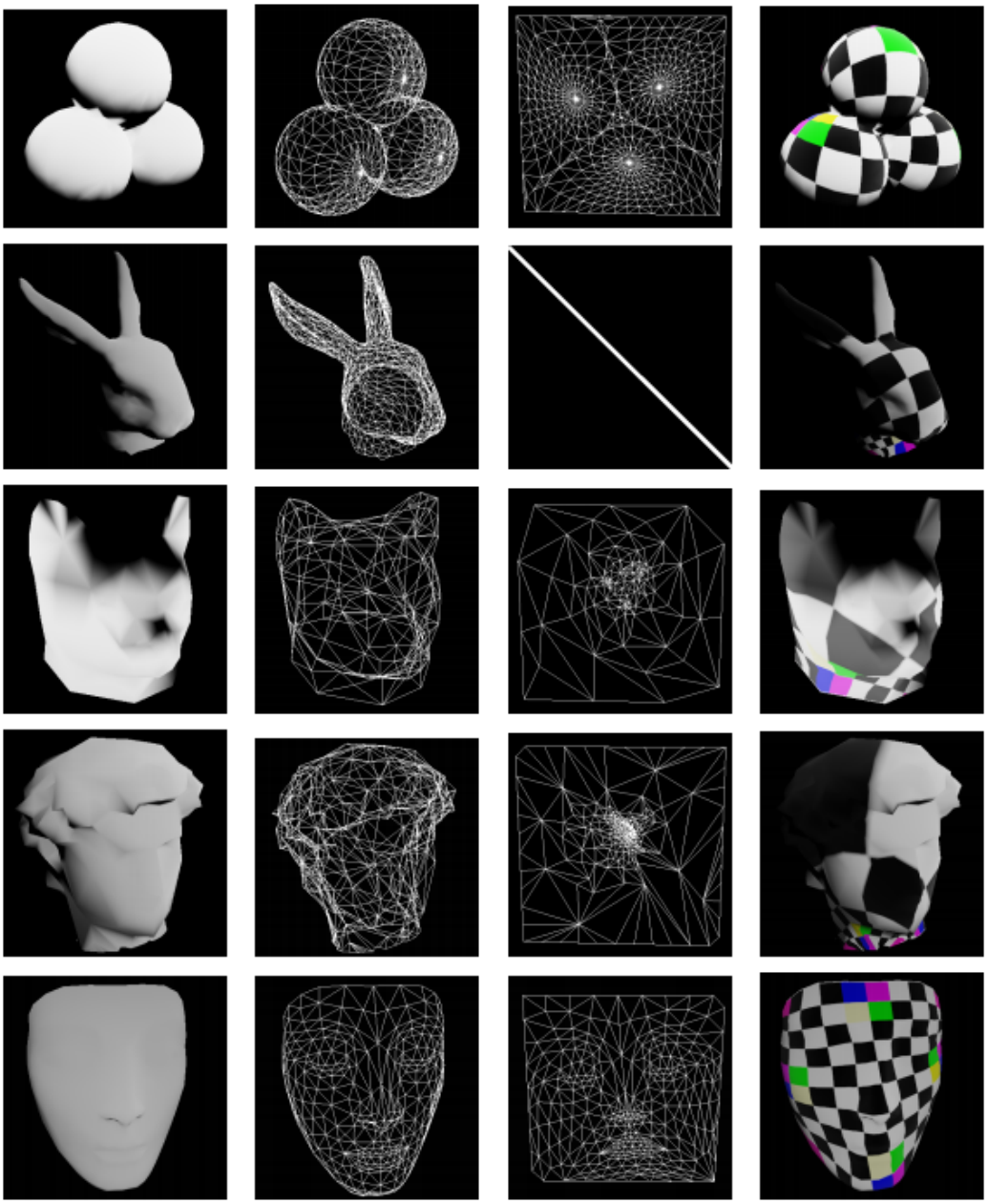

FIGURE 22 – HW4-cot-gptm-square

From left to right, it is an untexed map, a primitive triangular mesh structure, an average-weighted circle map and its corresponding texture map. The 2D texture is the colored checkerboard map used in ImageWarping.

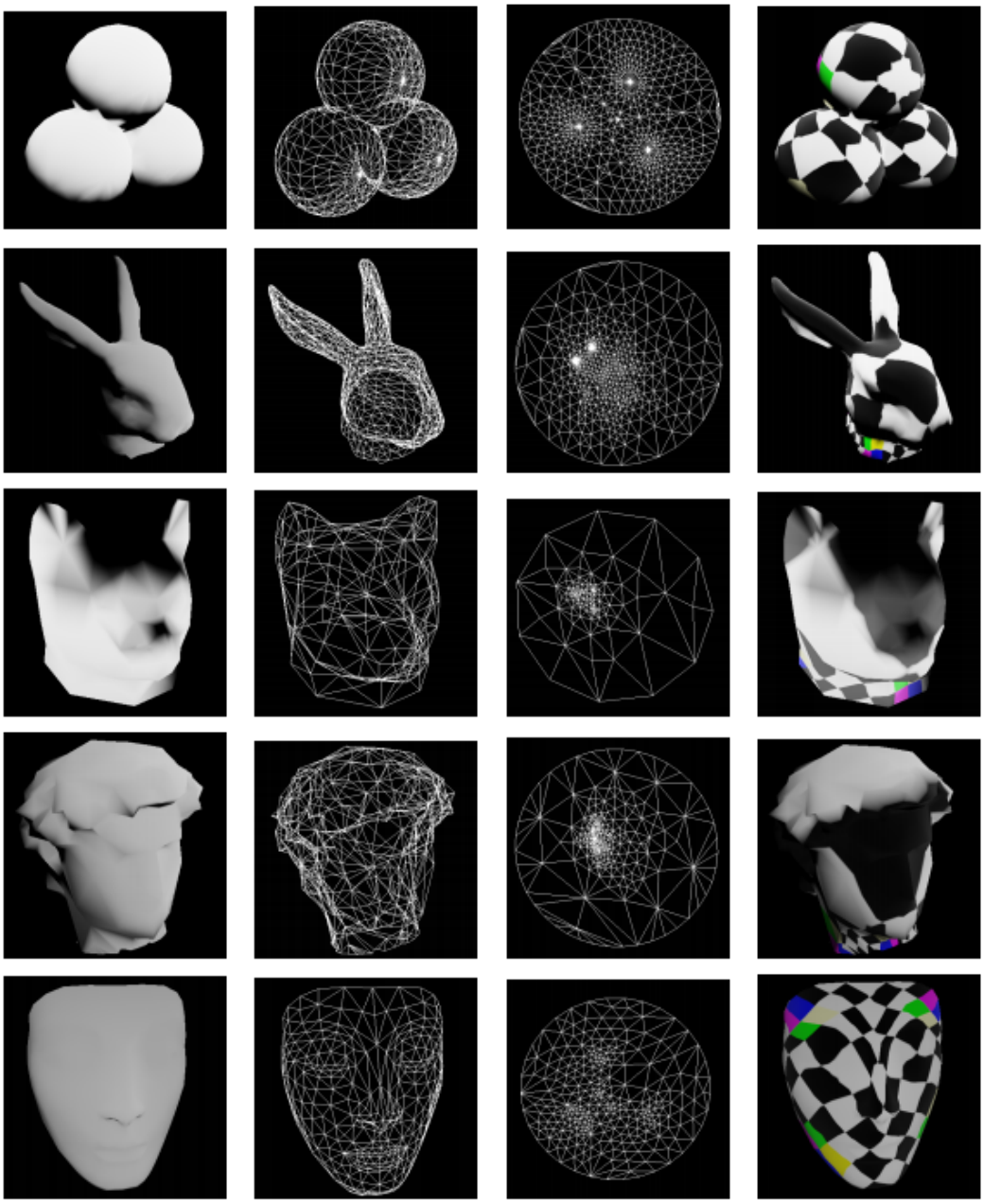

FIGURE 23 – HW4-uniform-gptm-circle

From left to right, it is an untexed map, a primitive triangular mesh structure, an cotangent-weighted circle map and its corresponding texture map. The 2D texture is the colored checkerboard map used in ImageWarping.

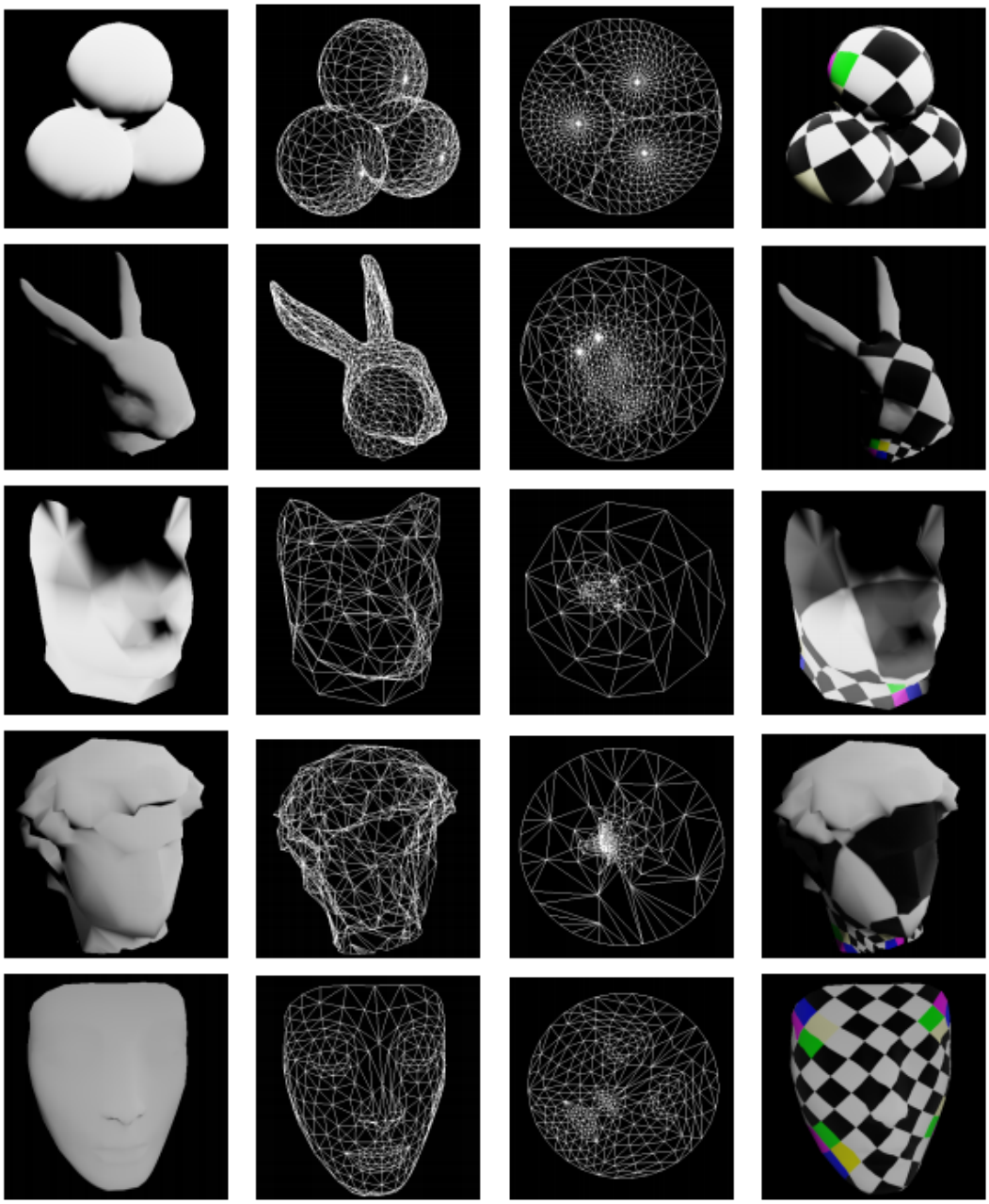

FIGURE 24 – HW4-cot-gptm-circle

3.2. ARAP

3.2.1. Requests

MOtivation

The parameterization method in $HW4^{3.1}$ is to fix the boundary, which maintains the parameterization result of valid (flip-free), but the deformation of the triangle is larger. If the boundary is not fixed, the points of the boundary also have degrees of freedom to move, which can reduce the shape variable of the triangle and thus obtain better parametric results.

1. ASAP First, implement the ASAP algorithm (As Similar As Possible) to become more familiar with the construction of mesh Laplacian matrices and solve sparse systems of equations
2. ARAP Then, implement the ARAP algorithm (As Rigid As Possible) Both algorithms come from the paper : [5]

3.2.2. Energy

Assume the triangles of the 3D triangle mesh are num-bered with t = 1 to T and the area of the 3D triangles are A_t . Assume that each 3D triangle is equipped with its own local isometric parameterization using a triangle in the plane $x_t = x_t^0, x_t^1, x_t^2$. Our goal is to find a single parameterization of the entire mesh, i.e., a piecewise linear mapping from the 3D mesh to the 2D plane, described by assigning 2D coordinates u to each of the n vertices. For triangle t, let us denote these 2D coordinates as $u_t = u_t^0, u_t^1, u_t^2$. Given this setup, the mapping between x_t and u_t has an associated 2 × 2 Jacobian matrix which is constant per triangle. We denote this matrix at triangle t as $J_t(u)$ to express its dependence on the u. It represents the linear portion of the affine mapping from the triangle described by x_t to the triangle described by u_t . In our method, we will also assign an auxiliary linear transformation (2×2 matrix) Lt to each triangle taken from some family of allowed transformations M (in particular, we will consider, in turn, M to be the similarity transformations, and later, the rotations).
Define the energy of the parameterization coordinates u and an auxiliary set of T linear transformations $L = \{L_1, .., L_T\}$ to be

$$E(u, L) = \sum_{t=1}^{T} A_t ||J_t(u) - L_t||_F^2 \tag{22}$$

Find the parameters to minimize the energy (u,L)

$$(u, L) = \arg\min_{(u,L)} E(u, L) \tag{23}$$

When optimizing, both u and L are all optimized parameters. What we ultimately need is u, as a parameterized result ; L was introduced to assist in the optimization process. We can obtain different parametric results with different properties by selecting forms of L, such as rotation, or affine and rotation. For different forms of L, the method of optimizing E(u,L) is also different

3.2.3. ASAP

As Similar As Possible, set the L in the form of $\begin{bmatrix} a & b \\ -b & a \end{bmatrix}$ which guarantee the transformation as similar as possible assimilarity transformation. It suffice to solve the problem :

$$E(u, L) = \frac{1}{2} \sum_{t=1}^{T} \sum_{i=0}^{2} \cot\left(\theta_t^{i+2}\right) \left\| \left(u_t^i - u_t^{i+1}\right) - L_t \left(x_t^i - x_t^{i+1}\right) \right\|^2 \tag{24}$$

It's exactly a sparse linear equation :L contains the unknown quantity $\{a_1, \ldots, a_T\}$ and $\{b_1, \ldots, b_T\}$. Pinkall and Polthier prove that the energy equation can be written as

$$\lambda L_t = \begin{bmatrix} a_t & b_t \\ -b_t & a_t \end{bmatrix}, \text{ let } x_t^i - x_t^{i+1} = \Delta x_t^i, \text{ let } E_t^i = \frac{1}{2}\cot\left(\theta_t^{i+2}\right)\left\|\left(u_t^i - u_t^{i+1}\right) - L_t\left(x_t^i - x_t^{i+1}\right)\right\|^2 \text{ leads to}$$

$$\begin{aligned} E(u,L) &= \sum_{t=1}^{T}\sum_{i=0}^{2} E_t^i \\ E_t^i &= \frac{1}{2}\cot\left(\theta_t^{i+2}\right)\left\|\left(\begin{bmatrix} u_t^{i,1} \\ u_t^{i,2} \end{bmatrix} - \begin{bmatrix} u_t^{i+1,1} \\ u_t^{i+1,2} \end{bmatrix}\right) - \begin{bmatrix} a_t & b_t \\ -b_t & a_t \end{bmatrix}\begin{bmatrix} \Delta x_t^{i,1} \\ \Delta x_t^{t,2} \end{bmatrix}\right\|^2 \\ &= \frac{1}{2}\cot\left(\theta_t^{i+2}\right)\left\{\left(u_t^{i,1} - u_t^{i+1,1} - a_t\Delta x_t^{i,1} - b_t\Delta x_t^{i,2}\right)^2\right. \\ &\left. + \left(u_t^{i,2} - u_t^{i+1,2} - a_t\Delta x_t^{i,2} + b_t\Delta x_t^{i,1}\right)^2\right\} = \frac{1}{2}\cot\left(\theta_t^{i+2}\right)\left(\left(Q_t^{i,1}\right)^2 + \left(Q_t^{i,2}\right)^2\right) \end{aligned} \tag{25}$$

where the $\theta_t^i denote the triangle x_t = \left\{x_t^0, x_t^1, x_t^2\right\}$ith angle. Where :

$$Q_t^{i,1} = u_t^{i,1} - u_t^{i+1,1} - a_t\Delta x_t^{i,1} - b_t\Delta x_t^{i,2} \tag{26}$$

$$Q_t^{i,2} = u_t^{i,2} - u_t^{i+1,2} - a_t\Delta x_t^{i,2} + b_t\Delta x_t^{i,1}$$

$$\begin{gathered} \frac{\partial E_t^i}{\partial u_t^{i,1}} = \cot\left(\theta_t^{i+2}\right)Q_t^{i,1}, \quad \frac{\partial E_t^i}{\partial u_t^{i+1,1}} = -\cot\left(\theta_t^{i+2}\right)Q_t^{i,1}, \quad \frac{\partial E_t^i}{\partial u_t^{i,2}} = \cot\left(\theta_t^{i+2}\right)Q_t^{i,2}, \\ \frac{\partial E_t^i}{\partial u_t^{i+1,2}} = -\cot\left(\theta_t^{i+2}\right)Q_t^{i,2}, \quad \frac{\partial E_t^i}{\partial a_t} = \cot\left(\theta_t^{i+2}\right)\left(-Q_t^{i,1}\Delta x_t^{i,1} - Q_t^{i,2}\Delta x_t^{i,2}\right), \\ \frac{\partial E_t^i}{\partial b_t} = \cot\left(\theta_t^{i+2}\right)\left(-Q_t^{i,1}\Delta x_t^{i,2} + Q_t^{i,2}\Delta x_t^{i,1}\right) \end{gathered} \tag{27}$$

To minimize $E(u,L)$, let $\nabla_{u,a,b}E(u,L) = \mathbf{0}$. Denote $N_T\left(u^{[k]}\right) = \left\{(i,t) : u^{[k]} \in u_t \text{as} u_t^i\right\}$ to be the triangle which contains the Common nodes $u^{[k]}$. We derive the follow equation :

$$\begin{aligned} 0 = \frac{\partial E}{\partial u^{[k],1}} &= \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial E_t}{\partial u_t^{i,1}} = \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial\left(E_t^i + E_t^{i+1} + \partial E_t^{i+2}\right)}{\partial u_t^{i,1}} \\ &= \sum_{(i,t)\in N_T\left(u^{[k]}\right]),} \cot\left(\theta_t^{i+2}\right)Q_t^{i,1} - \cot\left(\theta_t^{i+1}\right)Q_t^{i+2,1} \\ 0 = \frac{\partial E}{\partial u^{[k],2}} &= \sum_{(i,t)\in N_T\left(u^{[k]}\right]),} \frac{\partial E_t}{\partial u_t^{i,2}} = \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial\left(E_t^i + E_t^{i+1} + \partial E_t^{i+2}\right)}{\partial u_t^{i,2}} \\ &= \sum_{(i,t)\in N_T\left(u^{[k]}\right]),} \cot\left(\theta_t^{i+2}\right)Q_t^{i,2} - \cot\left(\theta_t^{i+1}\right)Q_t^{i+2,2} \\ 0 = \frac{\partial E(u,L)}{\partial a_t} &= \frac{\partial\left(E_t^0 + E_t^1 + \partial E_t^2\right)}{\partial a_t} = \sum_{i=0}^{2}\cot\left(\theta_t^{i+2}\right)\left(-Q_t^{i,1}\Delta x_t^{i,1} - Q_t^{i,2}\Delta x_t^{i,2}\right) \\ 0 = \frac{\partial E(u,L)}{\partial b_t} &= \frac{\partial\left(E_t^0 + E_t^1 + \partial E_t^2\right)}{\partial b_t} = \sum_{i=0}^{2}\cot\left(\theta_t^{i+2}\right)\left(-Q_t^{i,1}\Delta x_t^{i,2} + Q_t^{i,2}\Delta x_t^{i,1}\right) \end{aligned} \tag{28}$$

The paper states that ASAP may obtain a trivial solution (energy of 0) that can be fixed by selecting a needle point.

3.2.4. ARAP

As rigid as possible. Limit L to rotational transformations only, thus maintaining the size of the original mesh triangle as much as possible. Equivalent add one more constraint to L in ASAP $a^2 + b^2 = 1$. It can also be proved that the optimization of this is equivalent to minimization.

$$\sum_{t=1}^{T} A_t \left[\left(\sigma_{t,1} - 1\right)^2 + \left(\sigma_{t,2} - 1\right)^2 \right] \tag{29}$$

To solve the ARAP optimization problem, a local/global method is used. In the Local stage, the fixed parameterized point u, solving for the optimal L_t for each triangle; In the Global phase, fixed L_t solves for the optimal parameterization u. Iterate like this, and since each step convex energy function is decreasing, the iteration eventually converges. In the Global phase, solving the coefficient matrix of u is fixed, so we only need to decompose the matrix once, which can be reused in later iterations.

1. **Local Phase** For ARAP methods, you can use $S_t(u) = \sum_{i=0}^{2} \cot\left(\theta_t^i\right)\left(u_t^i - u_t^{i+1}\right)\left(x_t^i - x_t^{i+1}\right)^T$ instead of $J_t(u)$ for SVD decomposition to obtain$S_t(u) = U\Sigma V^T, \Sigma = \text{diag}\left(\sigma_{t,1}, \sigma_{t,2}\right)$ Then for $J_t(u)$ can minimize $\|_t(u) - L_t\|_F^2$ for $L_t = UV^T$, that is, the singular value becomes 1.
2. **Global Phase** When fixed L_t, $E(u, L)$ is a quadratic function about u, Setting the gradient to 0, we get $u^{[k]}$ are independent, and the coefficients share the same coefficient matrix) :

$$\begin{aligned}
0 = \frac{\partial E}{\partial u^{[k],1}} &= \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial E_t}{\partial u_t^{i,1}} = \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial\left(E_t^i + E_t^{i+1} + \partial E_t^{i+2}\right)}{\partial u_t^{i,1}} \\
&= \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \cot\left(\theta_t^{i+2}\right)\left(u_t^{i,1} - u_t^{i+1,1} - L_{t,11}\Delta x_t^{i,1} - L_{t,12}\Delta x_t^{i,2}\right) \\
&\quad - \cot\left(\theta_t^{i+1}\right)\left(u_t^{i+2,1} - u_t^{i,1} - L_{t,11}\Delta x_t^{i+2,1} - L_{t,12}\Delta x_t^{i+2,2}\right) \\
0 = \frac{\partial E}{\partial u^{[k],2}} &= \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial E_t}{\partial u_t^{i,2}} = \sum_{(i,t)\in N_T\left(u^{[k]}\right),} \frac{\partial\left(E_t^i + E_t^{i+1} + \partial E_t^{i+2}\right)}{\partial u_t^{i,2}} \\
&= \sum_{(i,t)\in N_T\left(u^{[k]}\right],} \cot\left(\theta_t^{i+2}\right)\left(u_t^{i,2} - u_t^{i+1,2} - L_{t,21}\Delta x_t^{i,1} - L_{t,22}\Delta x_t^{i,2}\right) \\
&\quad - \cot\left(\theta_t^{i+1}\right)\left(u_t^{i+2,2} - u_t^{i,2} - L_{t,21}\Delta x_t^{i+2,1} - L_{t,22}\Delta x_t^{i+2,2}\right)
\end{aligned} \tag{30}$$

3.2.5. Demo

Here is some demos about the ASAP method and ASAP. From the left to right, according to this is a prototriangular mesh structure mesh structure, the mesh structure of the object after "ASAP" parameterization, and the mesh structure of the object after "ARAP" parameterization. Note that in the ARAP, I set the iteration times 10, while it suffice to show satisfying results.
One can see that "ARAP" preserves more detail about the original object. since the transform matrix is set to be similar as a rotation matrix.

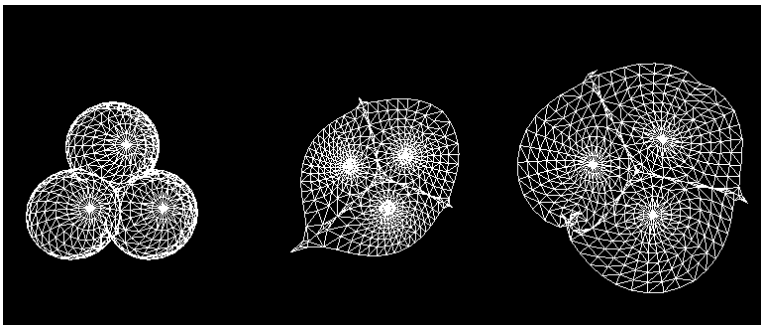

FIGURE 25 – HW5-demo-balls

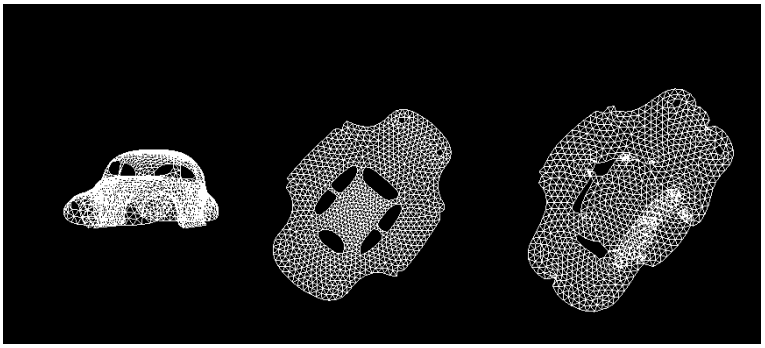

FIGURE 26 – HW5-demo-car

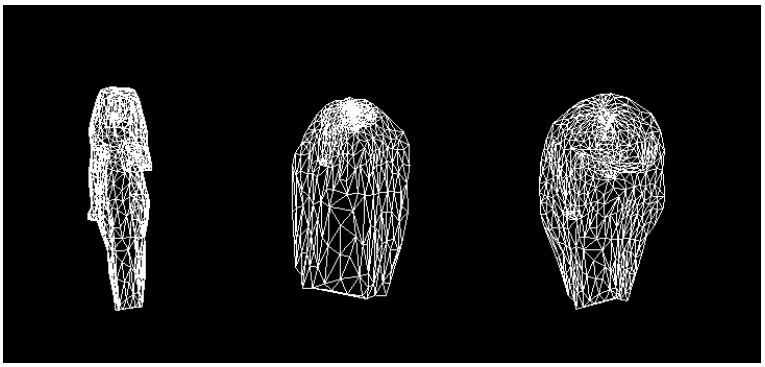

FIGURE 27 – HW5-demo-Isis

4. Simulation

4.1. MassSpring

4.1.1. Requests

Mass Spring System

Mass-spring systems provide a simple yet practical method for modeling a wide variety of objects, including cloth, hair, and deformable solids. However, as with other methods for modeling elasticity, obtaining realistic material behaviors typically requires constitutive parameters that result in numerically stiff systems. Explicit time integration methods are fast but when applied to these stiff systems they have stability problems and are prone to failure. Traditional methods for implicit

integration remain stable but require solving large systems of equations [Baraff and Witkin 1998; Press et al. 2007]. The high cost of solving these systems of equations limits their utility for real-time applications (e.g., games) and slows production work flows in off-line settings (e.g., film and visual effects).

1. Implement the accerlating algorithm in paper [6].
2. Compare two methods about to solve differential equation, namely, the implicit Euler method and the local-global method in the paper (note that we have apply such method in $HW5(ASAP)^{3.2}$).

4.1.2. Algorithm

The main problem in the mass spring system (in fact for almost all simulation problems) is : from the first n frame information, how can we get the $n+1$ frame information (displacement $\boldsymbol{x}$, speed $\boldsymbol{v}$) (set the time step to h)?

1. **Euler Implicit Method**

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h\boldsymbol{v}_{n+1}, \boldsymbol{v}_{n+1} = \boldsymbol{v}_n + h\boldsymbol{M}^{-1}(\boldsymbol{f}_{int}(t_{n+1}) + \boldsymbol{f}_{ext}), \tag{31}$$

let :

$$\boldsymbol{y} = \boldsymbol{x}_n + h\boldsymbol{v}_n + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_{ext} \tag{32}$$

Then the problem is transferred as solving the equation of x :

$$\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{M}(\boldsymbol{x} - \boldsymbol{y}) - h^2\boldsymbol{f}_{int}(\boldsymbol{x}) = 0, \tag{33}$$

Apply Newton iteration method, the main iteration part is listed as follow :

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - (\nabla\boldsymbol{g}(\boldsymbol{x}^{(k)}))^{-1}\boldsymbol{g}(\boldsymbol{x}^{(k)}). \tag{34}$$

Set the initial value of the iteration $\boldsymbol{x}^{(0)} = y$. After the iteration part, update the speed $v_{n+1} = (x_{n+1} - x_n)/h$ The above equation involves the derivation of elastic force. For a spring ($\boldsymbol{x}_1$,$\boldsymbol{x}_2$), the strength coefficient is k, the original length is l, we have :

$$\text{elastic force on } x_1 : \boldsymbol{f}_1(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(||\boldsymbol{x}_1 - \boldsymbol{x}_2|| - l)\frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{||\boldsymbol{x}_1 - \boldsymbol{x}_2||}, \text{ elastic force on } x_2 : \boldsymbol{f}_2(\boldsymbol{x}_1, \boldsymbol{x}_2) = -\boldsymbol{f}_1(\boldsymbol{x}_1, \boldsymbol{x}_2), \tag{35}$$

Calculate derivative of function $\boldsymbol{h}(\boldsymbol{x})$ with respect to $\boldsymbol{x}$, where $\boldsymbol{h}(\boldsymbol{x})$ is :

$$\boldsymbol{h}(\boldsymbol{x}) = k(||\boldsymbol{x}|| - l)\frac{-\boldsymbol{x}}{||\boldsymbol{x}||}, \tag{36}$$

we have :

$$\frac{d\boldsymbol{h}}{d\boldsymbol{x}} = k(\frac{l}{||\boldsymbol{x}||} - 1)\boldsymbol{I} - kl||\boldsymbol{x}||^{-3}\boldsymbol{x}\boldsymbol{x}^T. \tag{37}$$

Hence the elastic equation can be simplified as :

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{x}_1} = \frac{\partial \boldsymbol{h}(\boldsymbol{x}_1 - \boldsymbol{x}_2)}{\partial \boldsymbol{x}_1} = k(\frac{l}{||\boldsymbol{r}||} - 1)\boldsymbol{I} - kl||\boldsymbol{r}||^{-3}\boldsymbol{r}\boldsymbol{r}^T, \text{ where } \boldsymbol{r} = \boldsymbol{x}_1 - \boldsymbol{x}_2, \boldsymbol{I} \text{ is a Identity matrix.}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{x}_2} = -\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{x}_1}, \frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{x}_1} = -\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{x}_1}, \frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{x}_2} = \frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{x}_1}, \tag{38}$$

Derivatives matrix can be obtained by derivatives of all springs and assembling them (assembled as sparse matrices, matrices as symmetrical arrays)

2. **Acceleration Method**
In the above Explicit Euler method, Internal for (Conservative force), we have :

$$\boldsymbol{f}_{int}(x) = -\nabla E(\boldsymbol{x}) \tag{39}$$

Hence the problem (32) can be transferred as a optimization problem :

$$\frac{1}{2}k(||\boldsymbol{p}_1 - \boldsymbol{p}_2|| - r)^2 = \frac{1}{2}k \min_{||\boldsymbol{d}||=r} ||\boldsymbol{p}_1 - \boldsymbol{p}_2 - \boldsymbol{d}||^2, \tag{40}$$

which means we need to work out the problem :

$$\boldsymbol{x}_{n+1} = \min_{\boldsymbol{x},\boldsymbol{d}\in\boldsymbol{U}} \frac{1}{2}\boldsymbol{x}^T(\boldsymbol{M} + h^2\boldsymbol{L})\boldsymbol{x} - h^2\boldsymbol{x}^T\boldsymbol{J}\boldsymbol{d} - \boldsymbol{x}^T\boldsymbol{M}\boldsymbol{y} \tag{41}$$

where

$$\boldsymbol{U} = \{\boldsymbol{d} = (\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_s), \boldsymbol{d}_s \in R^3, ||\boldsymbol{d}_i|| = l_i\} \tag{42}$$

l_i refers to the initial length of the i^{th} spring. The matrices $\mathbf{L} \in \mathbb{R}^{3m\times 3m}$, $\mathbf{J} \in \mathbb{R}^{3m\times 3s}$ are defined as follows :

$$\mathbf{L} = \left(\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{A}_i^\top\right) \otimes \mathbf{I}_3, \mathbf{J} = \left(\sum_{i=1}^{s} k_i\mathbf{A}_i\mathbf{S}_i^\top\right) \otimes \mathbf{I}_3$$

where $\mathbf{A}_i \in \mathbb{R}^m$ is the incidence vector of i-th spring, i.e., $A_{i,i_1} = 1, A_{i,i_2} = -1$, and zero otherwise. Similarly, $\mathbf{S}_i \in \mathbb{R}^s$ is the i-th spring indicator, i.e., $S_{i,j} = \delta_{i,j}$. The matrix $\mathbf{I}_3 \in \mathbb{R}^{3\times 3}$ is the identity matrix and $\otimes$ denotes Kronecker product. Note that the matrix $\mathbf{L}$ is nothing but a stiffness-weighted Laplacian of the mass-spring system graph.
Alternate optimization on $\boldsymbol{x}$ and $\boldsymbol{d}$ gives the optimal solution :

(a) **Optimize x** Solve the equation : $(\boldsymbol{M} + h^2\boldsymbol{L})\boldsymbol{x} = h^2\boldsymbol{J}\boldsymbol{d} + \boldsymbol{M}\boldsymbol{y}$

(b) **Optimize d** $\boldsymbol{d}_i = l_i\frac{\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}}{||\boldsymbol{p}_{i_1} - \boldsymbol{p}_{i_2}||}$, where l_i refers to the initial length of spring, $\boldsymbol{p}_{i_1}\boldsymbol{p}_{i_2}$ are the two endpoints of the spring.

Iterating process will go on until the solution converges.

4.1.3. Demo

Here is some short video demos of the acceleration method. First, I test the acceleration method on a square grid, to simulate how the curtain interact with the blowing wind, both sparse (10×10) and dense grid(20×20) give a satisfied results.
Some parameters : for the grid I set the stiffness coefficient 10^5. While for the cuboid. I compare how the cuboid react when the only force is gravity setting the stiffness coefficient to be 10^3, 10^4, 10^5 each time. The follow pictures shows the final statement of the cuboid. As we can see, when the stiffness coefficient is set to be 10^4, 10^5, the stiffness coefficient is so strong that only Elastic deformation can be observed, nevertheless, if the stiffness coefficient is 10^3 or even smaller, the cuboid will collapse, since this time the dominant deformation is plastic deformation.
For more information, youcan download the demo videos here : HW6-demo-video

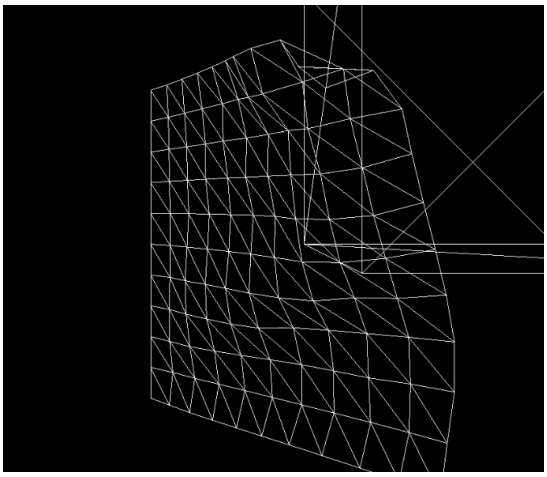

FIGURE 28 – HW6-grid-sparse

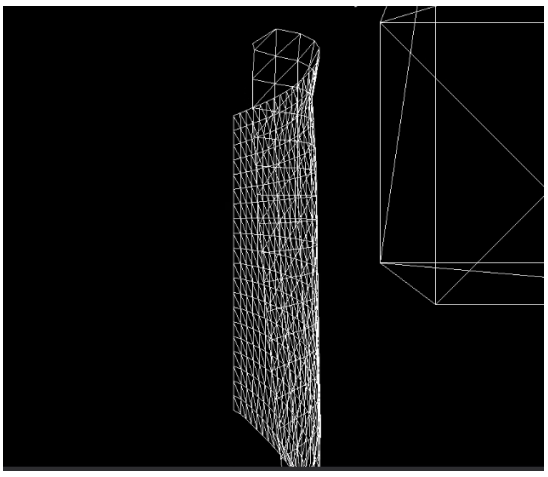

FIGURE 29 – HW6-grid-dense

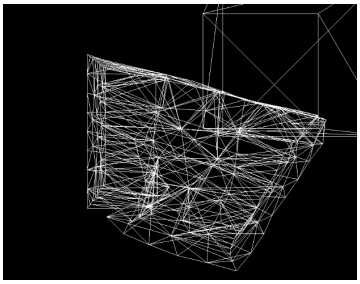

FIGURE 30 – HW6-cub-e3

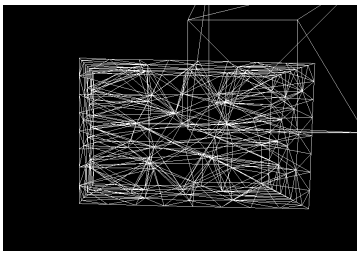

FIGURE 31 – HW6-cub-e4

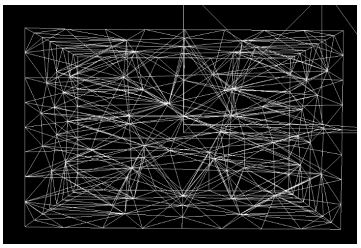

FIGURE 32 – HW6-cub-e5

4.2. SimulationTaichi

4.2.1. Requests

Taichi

Taichi is a parallel programming language for high-performance numerical computations. It is embedded in Python, and its just-in-time compiler offloads compute-intensive tasks to multi-core CPUs and massively parallel GPUs.Taichi-github

1. Create your own scene and objects using Taichi.
2. Change the parameters to see the difference of the impact with the objects' interaction
3. Have fun with to explore some new simluation results.

Some reference : [3] [4] [9]

4.2.2. Astronomy

As we all know, the universal gravitation equation :

$$F = \frac{GMm}{r^2} \tag{43}$$

where, G refers to the Gravitational constant, M and m are the mass of two objects. Moreover, we have the Kepler's law, which is listed as follow :

Kepler's laws–from WIKIPEDIA

Kepler's laws of planetary motion are three laws that describe the motion of planets around the sun :

1. Planets move around the sun in elliptic orbits. The sun is in one of the two foci of the orbit.
2. A line segment joining a planet and the Sun sweeps out equal areas during equal intervals of time.
3. The square of the orbital period of a planet is proportional to the cube of the semi-major axis of its orbit.

The trajectory of the celestial body is a conical curve (elliptic, parabolic and hyperbolic), where the elliptic curve orbit is a Kepler orbit and the escape parabolic and hyperbolic orbits are non-Kepler orbits. Non-Kepler orbits can be divided into single gravitational fields and multi-gravitational fields.

4.2.3. My own Scene

I constructed a small gravitational scene in which the red sphere in the center represents a large celestial body (say the sun) and the small green sphere represents the spaceship.

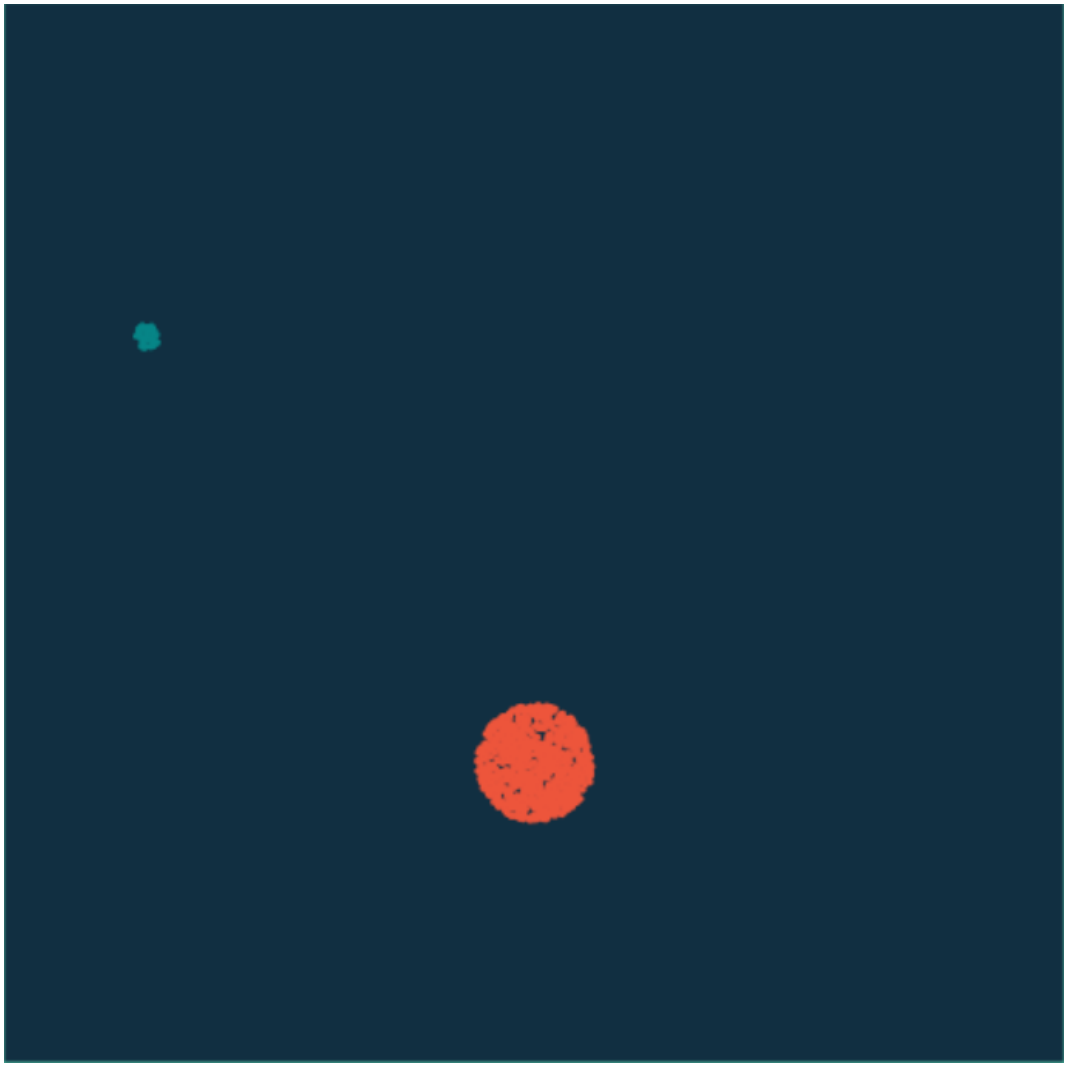

FIGURE 33 – HW7-demo-initial

Assuming that the mass of the central planet is much greater than the mass of the spacecraft and ignoring the autobiography of the planet, we can assume that the planet remains motionless throughout the process. This physical process is then a single point of mental strength.
From the he universal gravitation equation (43) and the Kepler's laws, we know that with fixed direction of the spacecraft and the initial position of both spacecraft and the planet, By changing the starting speed of the spaceship, spaceship can or can't escape from the gravitational field of the planet. For more information

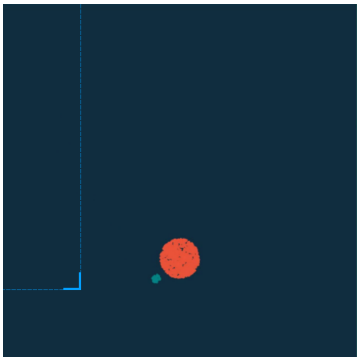

FIGURE 34 – HW7-catched

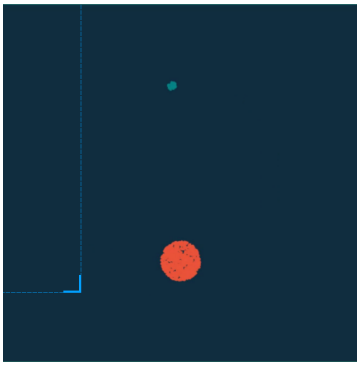

FIGURE 35 – HW7-critical-state

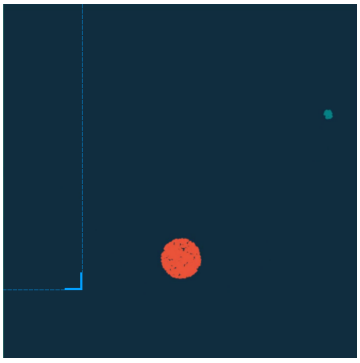

FIGURE 36 – HW7-escape

about the demo : HW7-video-demo

5. rendering

5.1. Shader

5.1.1. Requests

1. **Displacement mapping + Normal mapping**
2. **Denoise**
3. **Shadow mapping**

5.1.2. Normal Mapping and Displacement mapping

[2]The surfaces of objects are often not smooth, such as rusty iron, slightly undulating water, cloth, rough surfaces of wood.To make the rendering results more realistic, local features can be added through the mapping technique.Normal Mapping is a very economical method to change only local features by normal mapping. For example, in the 37. The blue [3] 2D texture is the normal map, some local features are added

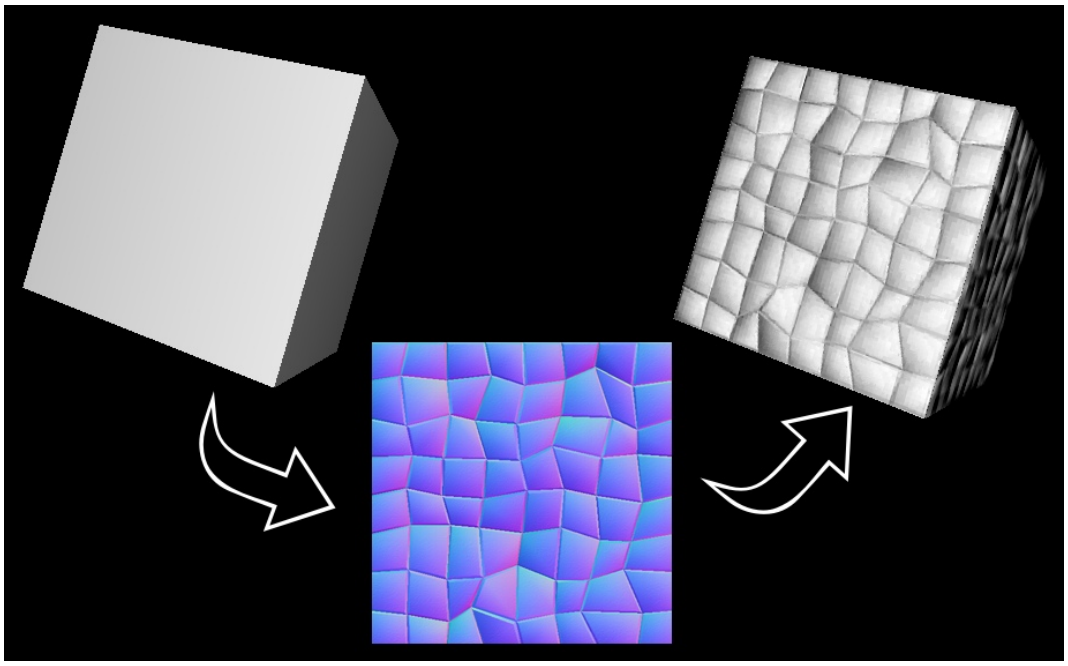

FIGURE 37 – HW8-request-shadow-mapping

through the mapping technique.
Displacement maps are used to change the position of vertices, representing the amount offset along the normal (0 is constant) Generally combined with the normal map to update the normal, get the correct rendering result.
The normal map does not change its position geometry, the surface is still flat, when exposed to reflect the larger surface bump or look from the side.The Displacement Mapping stores a height map that shifts the position of the surface along the normal direction (the original plane normal, not the normal map) to create a true bump.It needs to be used at the same time as the Normal Mapping technology, otherwise no light and shadow changes can be seen.

5.1.3. Mesh denoise

2. https ://learnopengl.com/Getting-started/Shaders
3. This is because the normal in the map is defined in the tangent space, with the up direction z-direction, corresponding to the B channel of the RGB

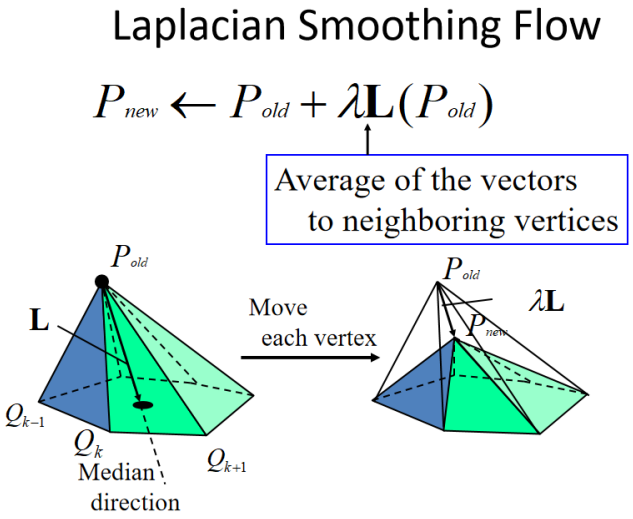

FIGURE 38 – HW8-request-mesh-denoise

Although the vertices are noiseed, the normals don't change, so the difference in rendering of the noisy model is mainly reflected in the distortion of the texture and the uneven edges. To achieve a good denoising effect, We only need to offset the vertices follow the algorithm mention in the 38. To list in details :

1. First, calculate the offset of each vertex :

$$\delta_i = p_i - \frac{1}{|N(i)|} \sum_{j \in N(i)} p_j \tag{44}$$

2. Secondly, Projects the offset onto a normal vector

$$\bar{\delta}_i = \langle \delta_i, \boldsymbol{n}_i \rangle \boldsymbol{n}_i \tag{45}$$

3. Then, Storage the value of $\langle \delta_i, \boldsymbol{n}_i \rangle$ into a displacement map.
4. Next, For other points in the texture map, the corresponding pixel value is interpolated by methods such as K nearby (K neighbor selected).
5. Finally, Scale all pixels' value in interval $[0, 1]$, the displacement map is generated

5.1.4. Shadow mapping

The basic idea of realizing the point light source shadow is to render the depth map from the perspective of the light source. To record the depth of the point closest to the light source in the corresponding texture coordinates in the source space. . In the process of chip element coloring, the vertex is transformed into the light source space through a transformation matrix to compare its depth and the corresponding depth map to determine whether the point is in the shadow. To list in details :

1. Create and render depth maps, calculate the transformation matrix of the world coordinates to the source space, the depth buffer is set by default when rendering ;
2. In a formal chip element shader, calculate the (perspective) coordinates of the corresponding vertex in the light source space, in order to sample from the depth map and compare with the depth map depth, the coordinate value of the coordinate is transformed to $[0, 1]$.
3. Compare the configuration depth sampled from the depth map and the depth of the point. If the point is farther, the value of "visibility" is set to be 0, otherwise, and the color of the chip element is calculated as a coefficient.
4. PCF method (Optional), which can bring about a softer shadow border.

5.1.5. Demo

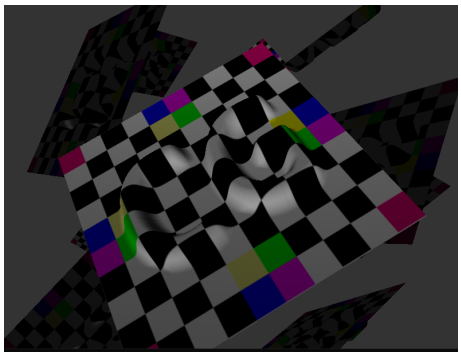

FIGURE 39 – HW8-nomal-mapping

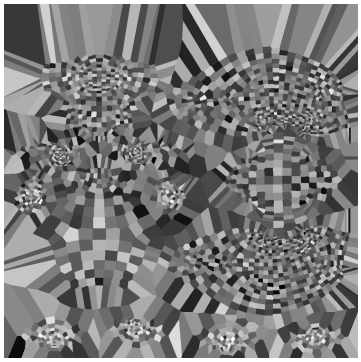

FIGURE 40 – HW8-denoise-displacement-map

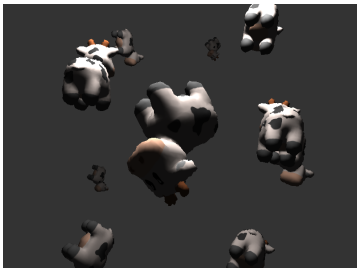

FIGURE 41 – HW8-noised

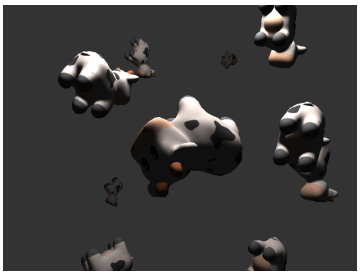

FIGURE 42 – HW8-denoised

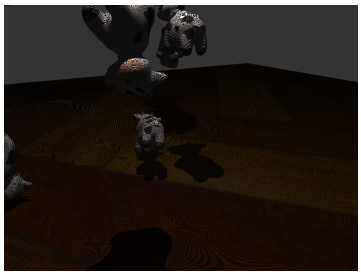

FIGURE 43 – HW8-shadow-original-map

FIGURE 44 – HW8-shadow-biased-map

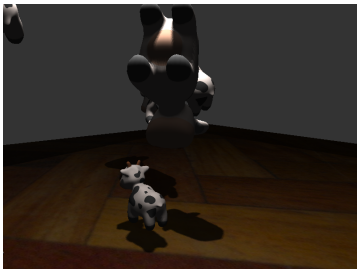

FIGURE 45 – HW8-shadow-PCF-map

5.2. Pathtracing

5.2.1. Requests

1. Implement the Path tracing algorithm
2. Ambient map importance sampling
3. Build your own scene and render it

5.2.2. Introduction

Rendering Equation

$$L_o(\boldsymbol{p},\boldsymbol{\omega}_o) = L_e(\boldsymbol{p},\boldsymbol{\omega}_o) + \int_{\mathcal{H}^2(\boldsymbol{n}(\boldsymbol{p}))} f_r(\boldsymbol{p},\boldsymbol{\omega}_i,\boldsymbol{\omega}_o) L_i(\boldsymbol{p},\boldsymbol{\omega}_i) \cos\theta_{\boldsymbol{\omega}_i,\boldsymbol{n}(\boldsymbol{p})} \boldsymbol{\omega}_i \tag{46}$$

— L_o Emitting light radiance
— $\boldsymbol{p}$ redering point
— $\boldsymbol{\omega}_i$ is the direction of the incoming light.
— $\boldsymbol{\omega}_o$ is the direction of the emitting light
— L_e is the emitting radiance
— $\boldsymbol{n}(\boldsymbol{p})$ is the normal vector of the point $\boldsymbol{p}$
— $\mathcal{H}^2(\boldsymbol{n}(\boldsymbol{p}))$ is the normal vector of the hemisphere where $\boldsymbol{n}(\boldsymbol{p})$ lies in.
— f_r is bidirectional scattering distribution function(BRDF)
— L_i is the radiance of the incoming light.
— $\theta_{\boldsymbol{\omega}_i,\boldsymbol{n}(\boldsymbol{p})}$ is the angle between $\boldsymbol{\omega}_i$ and $\boldsymbol{n}(\boldsymbol{p})$.

Denote :

$$\int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L = \int_{\mathcal{H}^2(\boldsymbol{n}(\boldsymbol{p}))} f_r(\boldsymbol{p},\boldsymbol{\omega}_i,\boldsymbol{\omega}_o) L \cos\theta_{\boldsymbol{\omega}_i,\boldsymbol{n}(\boldsymbol{p})} \boldsymbol{\omega}_i \tag{47}$$

Then :

$$L_o(\boldsymbol{p},\boldsymbol{\omega}_o) = L_e(\boldsymbol{p},\boldsymbol{\omega}_o) + \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_i(\boldsymbol{p},\boldsymbol{\omega}_i) \tag{48}$$

With reflecting equation :

$$L_r(\boldsymbol{p},\boldsymbol{\omega}_o) = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_i(\boldsymbol{p},\boldsymbol{\omega}_i) \tag{49}$$

For L_i there exist relationship :

$$L_i(\boldsymbol{p},\boldsymbol{\omega}_i) = L_o(\text{raytrace}(\boldsymbol{p},\boldsymbol{\omega}_i), -\boldsymbol{\omega}_i) \tag{50}$$

where raytrace is the ray's function of intersecting the scene raytrace$(\boldsymbol{p},\boldsymbol{\omega}_i)$ denote raytrace$(\boldsymbol{p},\boldsymbol{\omega}_i)$ as $\boldsymbol{p}'$, we have

$$L_i(\boldsymbol{p},\boldsymbol{\omega}_i) = L_o(\boldsymbol{p}', -\boldsymbol{\omega}_i) \tag{51}$$

Hence we form the iteration :

$$L_o(\boldsymbol{p},\boldsymbol{\omega}_o) = L_e(\boldsymbol{p},\boldsymbol{\omega}_o) + \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_o(\boldsymbol{p}', -\boldsymbol{\omega}_i) \tag{52}$$

A more efficient way to implement the iteration is to iterate on L_r

$$L_r(\boldsymbol{p},\boldsymbol{\omega}_o) = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} (L_e(\boldsymbol{p}', -\boldsymbol{\omega}_i) + L_r(\boldsymbol{p}', -\boldsymbol{\omega}_i)) \quad = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_e(\boldsymbol{p}', -\boldsymbol{\omega}_i) + \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_r(\boldsymbol{p}', -\boldsymbol{\omega}_i) \tag{53}$$

Define $L_{\text{dir}}(\boldsymbol{p},\boldsymbol{\omega}_o) = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_e(\boldsymbol{p}', -\boldsymbol{\omega}_i)$ as the direct light.
$L_{\text{indir}}(\boldsymbol{p},\boldsymbol{\omega}_o) = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_r(\boldsymbol{p}', -\boldsymbol{\omega}_i)$ as the indirect light.

5.2.3. Direct Light

$$L_{\text{dir}}(\boldsymbol{p},\boldsymbol{\omega}_o) = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_e(\boldsymbol{p}', -\boldsymbol{\omega}_i) \tag{54}$$

In the integral, for most directions $\boldsymbol{\omega}_i$, $L_e(\boldsymbol{p}',\boldsymbol{\omega}_i) = 0$ (non-light source), so we integrate directly in the direction where the light source is located. While $\boldsymbol{p}, \boldsymbol{\omega}_o, \boldsymbol{\omega}_i$ can be located by three points.

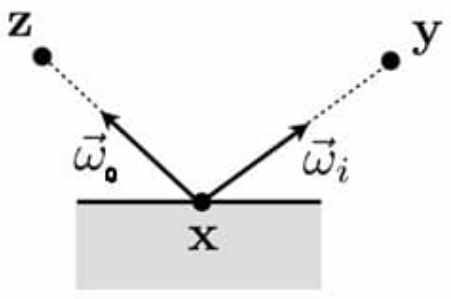

FIGURE 46 – HW9-direct-light

In the 46, $\boldsymbol{x}$ refers to $\boldsymbol{p}$; $\boldsymbol{y}$ refers to $\boldsymbol{p}'$ From geometry, we know that

$$\boldsymbol{\omega}_i = \frac{|\cos\theta_{\boldsymbol{y},\boldsymbol{x}}|}{\|\boldsymbol{x}-\boldsymbol{y}\|^2} A(\boldsymbol{y}) \tag{55}$$

where the $\theta_{\boldsymbol{y},\boldsymbol{x}}$ is the angle between $\boldsymbol{x}-\boldsymbol{y}$ and $\boldsymbol{n}(\boldsymbol{y})$. Introduction of geometric transfer terms ("transfer efficiency" between two points).

$$G(\boldsymbol{x}\leftrightarrow\boldsymbol{y}) = V(\boldsymbol{x}\leftrightarrow\boldsymbol{y})\frac{|\cos\theta_{\boldsymbol{x},\boldsymbol{y}}||\cos\theta_{\boldsymbol{y},\boldsymbol{x}}|}{\|\boldsymbol{x}-\boldsymbol{y}\|^2} \tag{56}$$

In the above equation, $V(\boldsymbol{x}\leftrightarrow\boldsymbol{y})$ is defined as visible function, which means its value is zero if there is no block between $\boldsymbol{x}$ and $\boldsymbol{y}$. It's obvious that G is a symmetric function, namely $G(\boldsymbol{x}\leftrightarrow\boldsymbol{y}) = G(\boldsymbol{y}\leftrightarrow\boldsymbol{x})$. Finally, we get

$$L_{\text{dir}}(\boldsymbol{x}\to\boldsymbol{z}) = \int_A f_r(\boldsymbol{y}\to\boldsymbol{x}\to\boldsymbol{z}) L_e(\boldsymbol{y}\to\boldsymbol{x}) G(\boldsymbol{x}\leftrightarrow\boldsymbol{y}) A(\boldsymbol{y}) \tag{57}$$

where the integration domain A is all the area in the scene, but only towards light source we have $L_e(\boldsymbol{y}\to\boldsymbol{x}) \neq 0$. If there are N_e light sources, and the light sources in the scene to be $\{L_{e_i}\}_{i=1}^{N_e}$. The corresponding region sets are $\{A(L_{e_i})\}_{i=1}^{N_e}$. equation (57) will be rewritten as :

$$L_{\text{dir}}(\boldsymbol{x}\to\boldsymbol{z}) = \sum_{i=1}^{N_e}\int_{A(L_{e_i})} f_r(\boldsymbol{y}\to\boldsymbol{x}\to\boldsymbol{z}) L_e(\boldsymbol{y}\to\boldsymbol{x}) G(\boldsymbol{x}\to\boldsymbol{y}) A(\boldsymbol{y}) \tag{58}$$

5.2.4. Indirect Light

The iteration form is :

$$L_r(\boldsymbol{p},\boldsymbol{\omega}_o) = \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_e(\boldsymbol{p}',-\boldsymbol{\omega}_i) + \int_{\boldsymbol{p},\boldsymbol{\omega}_o,\boldsymbol{\omega}_i} L_r(\boldsymbol{p}',-\boldsymbol{\omega}_i) \tag{59}$$

5.2.5. Important Sampling and Monte Carlo method

A function Y of a random variable X is also a random variable:

$$X \sim p(x)$$
$$Y = f(X)$$

Expected value of a function of a random variable:

$$E[Y] = E[f(X)] = \int f(x)\, p(x)\, dx$$

- **Goal:** Estimating $I = \int_a^b f(x)\, \mathrm{d}x$
- **Idea:** Constructing random variable $\langle I \rangle$
  - Such that $\mathbb{E}[\langle I \rangle] = I$
  - $\langle I \rangle$ is called an **unbiased estimator** of I

- Let $\langle I \rangle := \frac{f(X)}{p(X)}$, then: $\mathbb{E}[g(X)] = \int_a^b g(x)\, p(x)\, \mathrm{d}x$

$$\mathbb{E}[\langle I \rangle] = \mathbb{E}\left[\frac{f(X)}{p(X)}\right] = \int_a^b \frac{f(x)}{p(x)} p(x)\, \mathrm{d}x = I$$

- To estimate $\mathbb{E}[\langle I \rangle]$: strong law of large numbers
- Goal: to estimate $I = \int_a^b f(x)\, \mathrm{d}x$
  - Pick a probability density function $p(x)$
  - Generate *n* independent samples:
    $$x_1, x_2, \ldots, x_n \sim p$$
  - Evaluate $\hat{I}_j := \frac{f(x_j)}{p(x_j)}$ for *j* = 1, 2, …, *n*
  - Return sample mean: $\bar{I} := \frac{1}{n} \sum_{j=1}^{n} \hat{I}_j$

FIGURE 47 – HW9-imporatant-sampling

5.2.6. The calculation of the rendering equation

The integration we need to calculate is :

$$L_r(\boldsymbol{p}, \boldsymbol{\omega}_o) = L_{\text{dir}} + L_{\text{indir}} \tag{60}$$

Monte Carlo points are used to convert the points into samples :

$$\begin{aligned} L_{\text{dir}}(\boldsymbol{x} \to \boldsymbol{z}) &\approx \sum_{i=1}^{N_e} \sum_{j=1}^{N_i} \frac{f_r(\boldsymbol{y}_i^{(j)} \to \boldsymbol{x} \to \boldsymbol{z}) L_e(\boldsymbol{y}_i^{(j)} \to \boldsymbol{x}) G(\boldsymbol{x} \to \boldsymbol{y}_i^{(j)})}{p(\boldsymbol{y}_i^{(j)})} \\ L_{\text{indir}}(\boldsymbol{p}, \boldsymbol{\omega}_o) &\approx \sum_{k=1}^{N} \frac{f_r(\boldsymbol{p}, \boldsymbol{\omega}_i^{(k)}, \boldsymbol{\omega}_o) L_r(\boldsymbol{p}'^{(k)}, -\boldsymbol{\omega}) \cos\theta_{\boldsymbol{\omega}_i, \boldsymbol{n}(\boldsymbol{p})}}{p(\boldsymbol{\omega}_i^{(k)})} \end{aligned} \tag{61}$$

L_{dir} is sampled among all light sources. While we sample in the hemisphere for L_{indir}. We only sample once ($N_i = 1 \quad (i = 1, \ldots, N_e), N = 1$)

5.2.7. Ambient map importance sampling

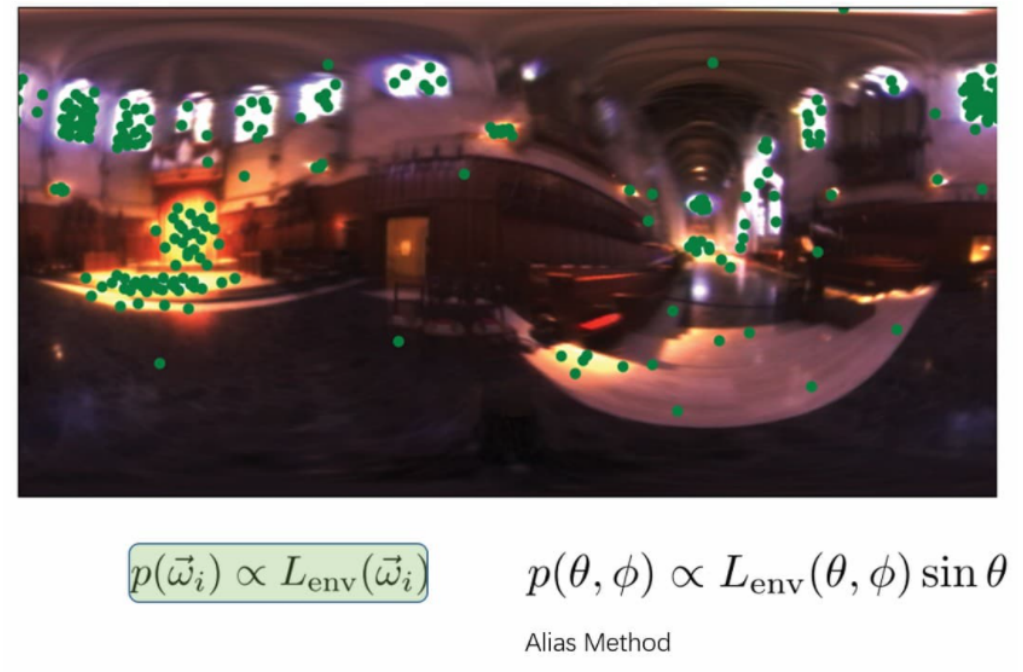

FIGURE 48 – HW9-Ambient-map-importance sampling

I use Alias method here Alias, with time complexity $O(1)$ After generating the probability table and the alias table, you can sample discrete pixels, and the correlation probability relationship is as follows :

$$
\begin{aligned}
1 = \int_I p_{\text{img}}(i,j) ij &= \int_\Theta p_{\text{img}}(\theta,\phi) \left|\frac{\partial(i,j)}{\partial(\theta,\phi)}\right| \theta\phi \\
&= \int_A p_{\text{img}}(A) \left|\det J_A\Theta\right| \left|\frac{\partial(i,j)}{\partial(\theta,\phi)}\right| A \\
&= \int_\Omega p_{\text{img}}(\boldsymbol{\omega}_i) \left|\frac{\mathrm{d}A}{\mathrm{d}\boldsymbol{\omega}_i}\right| \left|\det J_A\Theta\right| \left|\frac{\partial(i,j)}{\partial(\theta,\phi)}\right| \boldsymbol{\omega}_i \\
&= \int_\Omega p(\boldsymbol{\omega}_i)\boldsymbol{\omega}_i
\end{aligned}
\tag{62}
$$

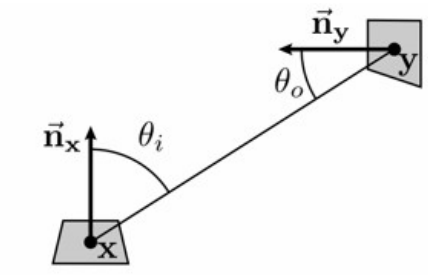

FIGURE 49 – HW9-dwi-dA

$$\left|\frac{\mathrm{d}\boldsymbol{\omega}_i}{\mathrm{d}A}\right| = \frac{|\cos\theta_o|}{\|\boldsymbol{x}-\boldsymbol{y}\|^2} = \frac{1}{R^2}$$
$$|\det J_A\Theta| = \frac{1}{R^2\sin\theta} \tag{63}$$
$$\left|\frac{\partial(i,j)}{\partial(\theta,\phi)}\right| = \frac{wh}{2\pi^2}$$

Hence we have :

$$p(\boldsymbol{\omega}_i) = \frac{wh}{2\pi^2\sin\theta}p_{\text{img}}(i,j) \tag{64}$$

Parameters

— $i \in [0,w]$, w is the with of the image.
— $j \in [0,h]$, h is the height of the image.
— $u,v \in [0,1]$, $u,v \in [0,1]$, where $u = i/w$ and $v = j/h$
— $\theta \in [0,\pi]$, $\theta = \pi(1-v)$
— $\phi \in [0,2\pi]$, $\phi = 2\pi u$
— $\boldsymbol{\omega}_i = (\sin\theta\sin\phi, \cos\theta, \sin\theta\cos\phi)$

5.2.8. Demo

The first two parts is the scene of the Cornell Box :
The first part is a comparison of rendered image whether using important sampling.
The second part is a comparison of rendered image whether using high ssp (the number of rays emitting from a single pixel).
The last part is my own scene, I create a night scene with a Gargoyle model.

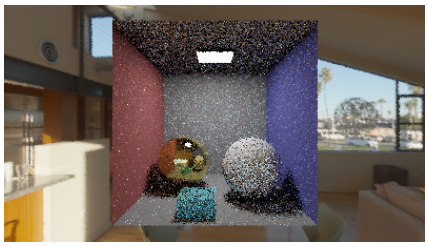

FIGURE 50 – HW9-no-important-sampling-ssp=2

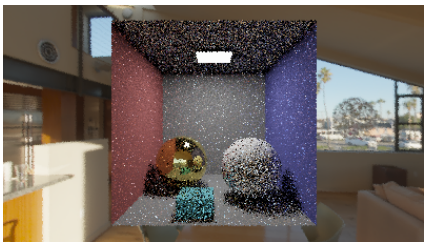

FIGURE 51 – HW9-important-sampling-ssp=2

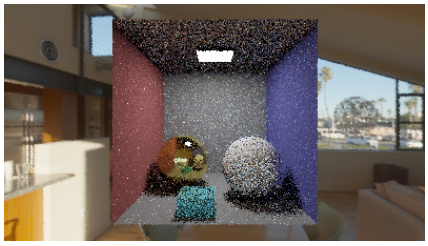

FIGURE 52 – HW9-no-important-sampling-ssp=2

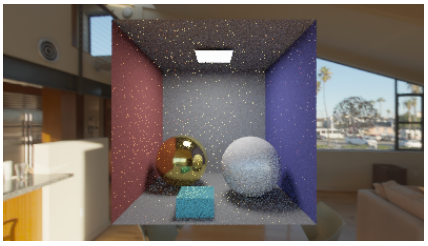

FIGURE 53 – HW9-no-important-samplimg-ssp=64

FIGURE 54 – HW9-my-scene-nolight

FIGURE 55 – HW9-my-scene-withlight

6. Thanks

I would like to thank Mr. Liu for his careful guidance to me, the procedural framework that the teaching assistants have worked hard to set up, and the seniors who have helped me

Références

[1] Nur Arad and Daniel Reisfeld. Image warping using few anchor points and radial functions. In *Computer graphics forum*, volume 14, pages 35–46. Wiley Online Library, 1995.

[2] Michael S Floater. Parametrization and smooth approximation of surface triangulations. *Computer aided geometric design*, 14(3) :231–250, 1997.

[3] Yuanming Hu, Yu Fang, Ziheng Ge, Ziyin Qu, Yixin Zhu, Andre Pradhana, and Chenfanfu Jiang. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling. *ACM Transactions on Graphics (TOG)*, 37(4) :1–14, 2018.

[4] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi : a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6) :1–16, 2019.

[5] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J Gortler. A local/global approach to mesh parameterization. In *Computer Graphics Forum*, volume 27, pages 1495–1504. Wiley Online Library, 2008.

[6] Tiantian Liu, Adam W Bargteil, James F O'Brien, and Ladislav Kavan. Fast simulation of mass-spring systems. *ACM Transactions on Graphics (TOG)*, 32(6) :1–7, 2013.

[7] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM SIGGRAPH 2003 Papers*, pages 313–318. 2003.

[8] D. Ruprecht and H. Müller. Image warping with scattered data interpolation methods. 1995.

[9] Alexey Stomakhin, Craig Schroeder, Lawrence Chai, Joseph Teran, and Andrew Selle. A material point method for snow simulation. *ACM Transactions on Graphics (TOG)*, 32(4) :1–10, 2013.