Mesh Simplification

The mesh simplification algorithm is implemented using python (please see the python files: mesh_simplify.py, class_mesh_simplify.py, and class_3d_model.py).

a) A brief introduction of the mesh simplification algorithm used in this assignment

There have been various kinds of mesh simplification algorithms. In the assignment, I use the simplification algorithm of [1] which is built on vertex pair contractions and error quadrics. The followings are the summary of the algorithm:

1. Calculate Q matrices for each vertex $v_i = [x_{v_i}, y_{v_i}, z_{v_i}, 1]^T$.

Each vertex v_i is the intersection of a set of planes $planes(v_i)$. A plane p can be represented as $p = [a, b, c, d]^T$ (defined by the equation $ax + by + cz + d = 0$ where $a^2 + b^2 + c^2 = 1$). Then Q_i (for vertex v_i) is defined as follows:

$$Q_i = \sum_{p \in planes(v_i)} pp^T$$

2. Select valid vertex pairs.

A vertex pair (v_i, v_j) is valid for contraction if either:
(v_i, v_j) is an edge, or $||v_i, v_j|| < t$ where t is a given threshold parameter.

3. Calculate the optimal contraction vertex v_{opt} which minimizes the error $cost_{ij}$ for each valid pair (v_i, v_j).

First, set $Q_{opt} = Q_i + Q_j = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{32} & q_{33} & q_{34} \\ q_{14} & q_{24} & q_{34} & q_{44} \end{bmatrix}$.

Define the $cost_{ij}$ as follows:

$$cost_{ij} = v^T Q_{opt} v$$

We want to find a solution v_{opt} to minimize the $cost_{ij}$. v_{opt} can be calculated as follows:

$$v_{opt} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{12} & q_{22} & q_{23} & q_{24} \\ q_{13} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

If the matrix is not invertible. We find the v_{opt} by assigning v_{opt} to be v_i, v_j, and the middle point of v_i and v_j to see whose $cost_{ij}$ is the minimum.

4. Place all the valid pairs in the order of $cost_{ij}$ with the minimum $cost_{ij}$ pair on the top.
5. Remove the minimum $cost_{ij}$ pair, contract (v_i, v_j) to be v_{opt}, and update the $cost_{ij}$ of all valid pairs involving v_i or v_j.

b) Instruction

System requirement: python 3.7, numpy, and argparse.

How to run the program

You can run the program by typing:

python mesh_simplify.py -i [input file path] -o [output file path] -r [simplification ratio (0~1)] -t [threshold parameter]

For example, run:

python mesh_simplify.py -i models/dinosaur.obj -o dinosaur_simp_30_0.obj -r 0.3 -t 0

c) Implementation details

There are three python files: **mesh_simplify.py**, **class_mesh_simplify.py**, and **class_3d_model.py**.

Python file **class_3d_model.py** defines a class named **a_3d_model** which reads 3d model file and calculate plane equations for each face and Q matrices for each vertex. Seven variables are defined in the class (here, point and vertex are same terms):

1. **points**:
   a numpy.array, each row is a xyz coordinate of a vertex, shape: number_of_points * 3
2. **faces**:
   a numpy.array, each row represents a triangle consisting of three serial numbers of points (1~number_of_points), shape: number_of_faces * 3
3. **edges**:
   a numpy.array, each row represents an edge consisting of two serial numbers of points (1~number_of_points)
4. **number_of_points**:
   an int, number of points (vertices)
5. **number_of_faces**:

an int, number of faces

6. **plane_equ_para**:
   a numpy.array, each row consists of four plane equation parameters $[a, b, c, d]^T$, shape: number_of_faces * 4
7. **Q_matrices**:
   a list, each item is a 4*4 Q matrix (numpy.array), length of the list: number_of_points

There are three functions defined in the class, namely:

1. **load_obj_file()**:
   generate points, faces, edges, number_of_points, and number_of_faces;
2. **calculate_plane_equations()**:
   calculate plane_equ_para;
3. **calculate_Q_matrices()**:
   calculate Q_matrices.

Python file **class_mesh_simplify.py** defines a class named **mesh_simplify** which inherits **a_3d_model**. The class **mesh_simplify** is used to implement mesh simplification. The followings are 11 important variables defined in this class (please see the comments in the codes for implementation details):

1. **t**:
   threshold parameter;
2. **ratio**:
   simplification ratio;
3. **dist_pairs**:
   a numpy.array, to store valid pairs of $||v_i, v_j|| < t$;
4. **valid_pairs**:
   a numpy.array, each row is a pair of vertices;
5. **v_optimal**:
   a numpy.array, each row is the xyz coordinate of a v_{opt} corresponding to the same row (v_i, v_j) in valid_pairs;
6. **cost**:
   a numpy.array, each row is the $cost_{ij}$ value corresponding to the same row (v_i, v_j) in valid_pairs;
7. **new_point**:
   a numpy.array, shape: (3,), the xyz coordinate of current v_{opt};
8. **new_valid_pair**:
   a numpy.array, shape: (2,), represents current valid pair;
9. **status_points**:
   a numpy.array, shape: (number_of_poins,), for each position, 0 means no change, -1 means the point is deleted;
10. **status_faces**:
    a numpy.array, shape: (number_of_faces,), for each position, 0 means no change, -1

means the face is deleted;

11. **new_point_count**:
    an int, count the number of pair contractions v_{opt}.

There are 10 functions defined in the class:

1. **generate_valid_pairs()**:
   select all valid pairs;
2. **calculate_optimal_contraction_pairs_and_cost ()**:
   compute the optimal contraction target v_{opt} for each valid pair (v_i, v_j);
   the error ${v_{opt}}^T Q_{opt} v_{opt}$ of this target vertex becomes the cost of contracting that pair;
   place all the pairs in a heap keyed on cost with the minimum cost pair at the top;
3. **iteratively_remove_least_cost_valid_pairs()**:
   Iteratively remove the pair (v_i, v_j) of least cost from the heap;
   contract this pair, and update the costs of all valid pairs involving (v_i, v_j);
   until existing points = ratio * original points;
4. **calculate_plane_equation_for_one_face()**,
   update_plane_equation_parameters(),
   update_Q(),
   update_valid_pairs_v_optimal_and_cost(),
   and **update_optimal_contraction_pairs_and_cost()**
   are sub functions of iteratively_remove_least_cost_valid_pairs();
5. **generate_new_3d_model()**:
   generate the simplified 3d model (vertices, faces);
6. **output(output_filepath)**:
   output the model to output_filepath.

Python file **mesh_simplify.py** uses argparse to run the program.

d) Results and evaluation

Use the program to simplify two 3D models in ./models: **bunny.obj** (35292 vertices and 70580 faces) and **dinosaur.obj** (2002 vertices and 4000 faces).

For **dinosaur.obj**, set threshold parameter = 1, change simplification ratio across 50%, 30%, 10%, 5%, 1%. Please find the simplified models in ./results. The results are shown in Figure 1:

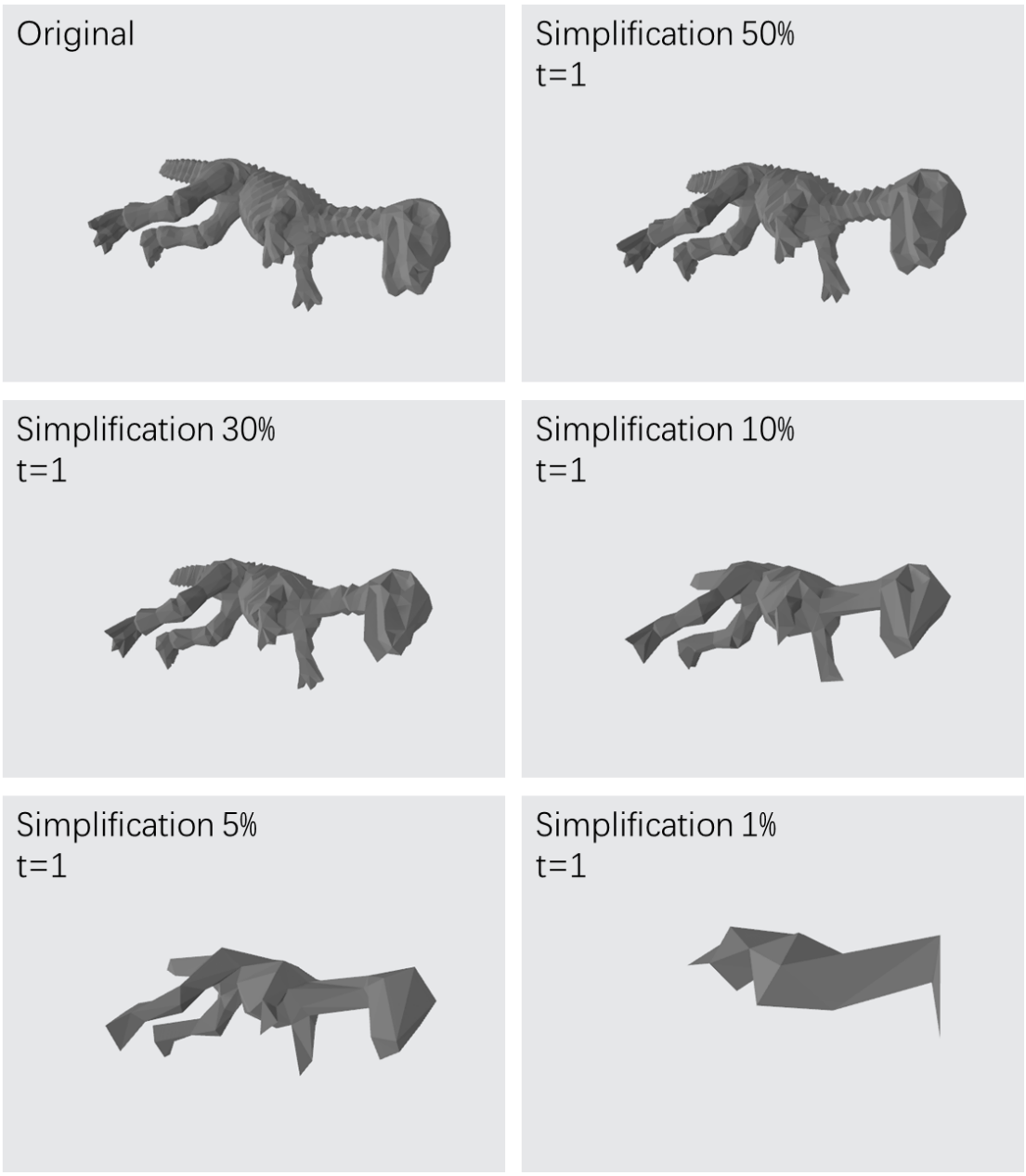

Figure 1

Table 1 shows the numbers of faces and vertices for each level of simplification.

Table 1

Name	Simplification level	Vertices	Faces
dinosaur.obj	None	2002	4000
dinosaur_simp_50_1.obj	50%	1000	1996
dinosaur_simp_30_1.obj	30%	599	1194
dinosaur_simp_10_1.obj	10%	197	390
dinosaur_simp_05_1.obj	5%	98	192
dinosaur_simp_01_1.obj	1%	19	32

From Figure 1 and Table 1 we can see that the program successfully simplifies the 3D model by simplifying the details while keep the main structure as much as possible. Even the simplification level is 5% with only 98 vertices and 192 faces remaining in the model, we can still recognize its shape. On the other hand, dinosaur_simp_50_1.obj shows that with this algorithm, the model looks the same even after removing about half of the vertices and faces.

For **bunny.obj**, set threshold parameter = 0, change simplification ratio across 50%, 30%, 10%, 1%. Please find the simplified models in ./results. The results are shown in Figure 2:

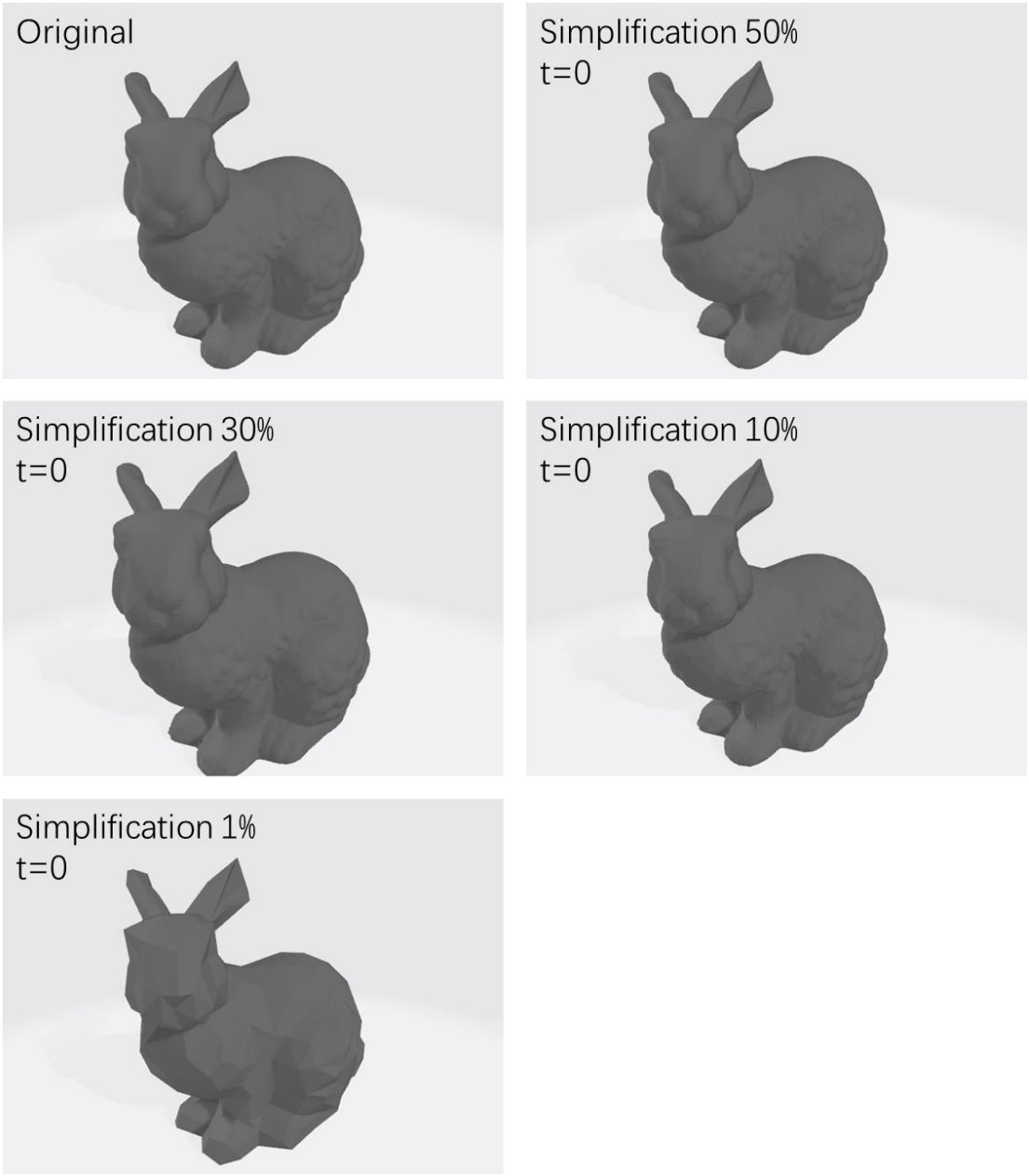

Figure 2

Table 2 shows the numbers of faces and vertices for each level of simplification.

Table 2

Name	Simplification level	Vertices	Faces
bunny.obj	None	35292	70580
bunny_simp_50_0.obj	50%	17645	35286
bunny_simp_30_0.obj	30%	10586	21168
bunny_simp_10_0.obj	10%	3493	6982
bunny_simp_01_0.obj	1%	349	694

From simplification level of 50% to 10%, we can see that the surface of the model becomes more and more rough. When the simplification level reaches 1%, the model becomes an abstract version of a bunny.

The influence of t value is tested by setting different threshold parameters (0, 1, and 10) with a fix simplification ratio 50% to simplify **dinosaur.obj**.

The appearances of the simplified models are almost the same. But the time costs are very different. Please see Figure 3 and Table 3.

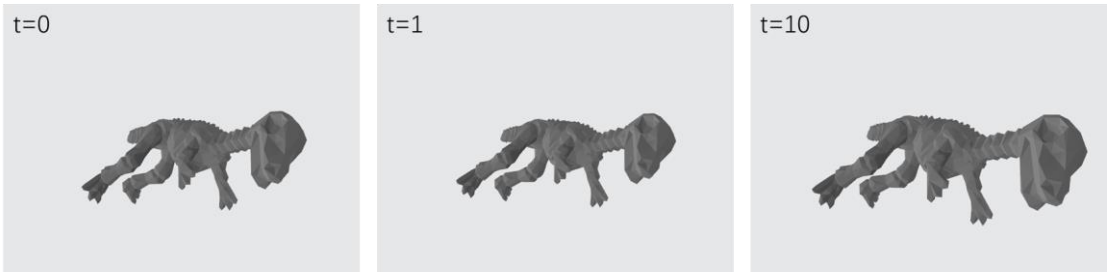

Figure 3

Table 3

Name	Simplification level	Time cost
dinosaur_simp_50_0.obj	50%	3.73 s
dinosaur_simp_50_1.obj	50%	3.73s
dinosaur_simp_50_10.obj	50%	54.02 s

We can infer that when setting threshold parameter = 10, there will be too many 'useless' valid pairs which cost much time but have no contribution to the final result. Threshold parameter is set by user. We need to be very careful when setting threshold.

Reference

[1] Garland, Michael, and Paul S. Heckbert. "Surface simplification using quadric error metrics." Proceedings of the 24th annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 1997.