Low-poly Mesh Generation for Building Models

Abstract

As a common practice, game modelers manually craft low-poly meshes for given 3D building models in order to achieve the ideal balance between the small element count and the visual similarity. This can take hours and involve tedious trial and error. This paper propose a novel and simple algorithm to automate this process by converting high-poly 3D building models into both simple and visually preserving low-poly meshes.

Keywords: mesh simplification, low-poly mesh, building model.

1 Introduction

In many applications the need for an accurate simplification of surface meshes is becoming more and more urgent. Mesh simplification can be used not only for rendering acceleration , but also for subsequent calculation tasks.

This article focuses on building model simplification , dedicates to solving the problem of how to automatically calculate the low mesh model that is almost consistent with the geometric appearance of the model given a large number of faces.

To solve this problem, this article propose a robust method for effectively generating extremely low-poly meshes that can be used as the coarsest level.And this paper first defines a visual metric to quantitatively measure the visual difference between the low-poly and the high-poly meshes,which guides how to simplify the input mesh.

2 Related works

2.1 Traditional mesh simplification

Traditional simplification algorithms repeatedly decimate the input mesh according to a cost function to preserve its rendered appearance, until the desired simplification ratio is reached. Simplification methods can be distinguished in two major categories: vertex clustering/decimation and edge collapse methods.

2.2 Planar-based mesh simplification for Building models

First, all the planes of the building model surface are detected, then the space is divided by the plane, and finally the model surface is extracted by optimization algorithm.

3 Method

3.1 Overview

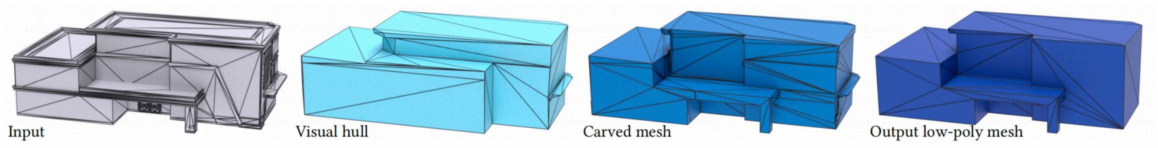


Figure 1: Overview of the method

As shown in Fig 1, we first construct a coarse visual hull by intersecting a small set of 3D primitives selected greedily to minimize our visual metric. These primitives are generated by computing and analyzing silhouettes of the input from a number of view directions. The result of the first stage is denoted as visual hull, which captures the input's silhouette but can miss important concave features. Our second stage generates a carved mesh from the visual hull by subtracting redundant volumes to recover concave features. We again deploy a greedy strategy to select the craving primitives by minimizing the visual metrics between the carved mesh and the input. Final stage derives the low-poly mesh by progressively applying edge collapse and edge-flip to the carved mesh.

4 Implementation details

4.1 Comparing with released source codes

No related source codes are available.

4.2 Experimental environment setup

C++ 11,CGAL,libigl.

4.3 Interface design

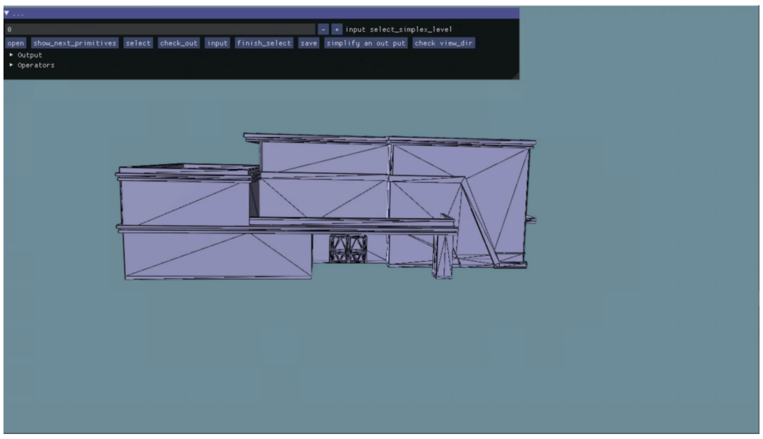


Figure 2: Interface

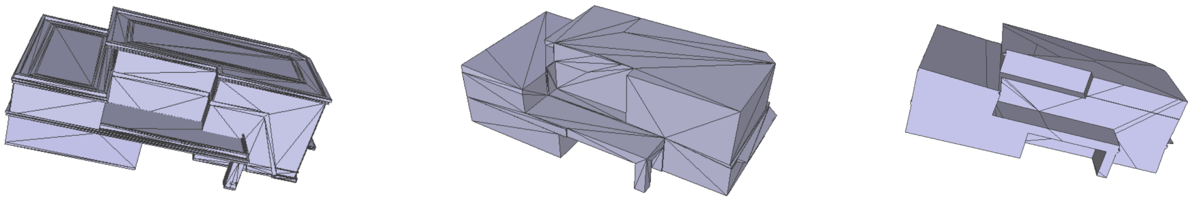
4.4 My contributions

1.Reproduce code.

2.Improve the second stage of the algorithm.

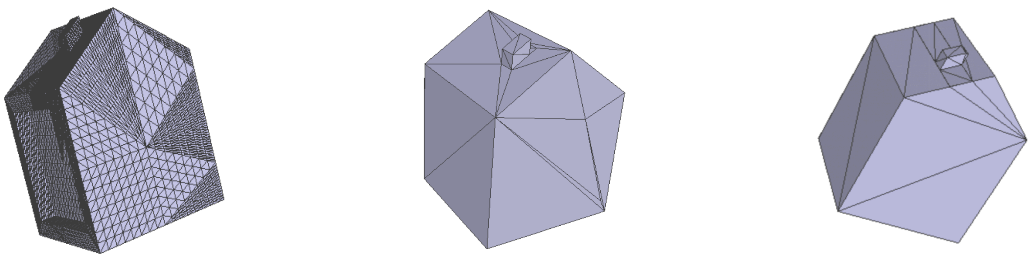
In the second stage,need a numerical value to select which plane to use for carving , Unlike the original text, which only uses visual error for selection,my implementation not only measuring the change of the visual error after the carve mesh, but also considering the size of the carve mesh's volume. When the visual error changes slightly, it is guided by the larger volume of the carve mesh, while ensuring that the visual error is basically unchanged.This optimization ensures that a concise model can still be obtained without many views.

5 Results and analysis



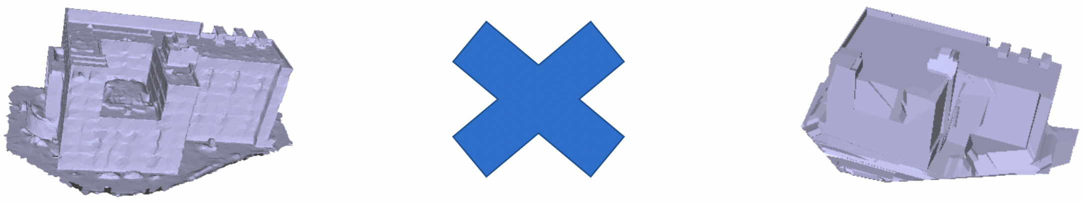
	Input	O	M
N of F	5502	246	326
N of P	3497	132	163
Time (s)		62s	38s

Figure 3: Experimental results



	Input	O	M
N of F	21612	52	60
N of P	10809	29	31
Time (s)		20s	75s

Figure 4: Experimental results



	Input	O	M
N of F	52256	Fail(>20min)	2932
N of P	26289	Fail(>20min)	1821
Time (s)			466s

Figure 5: Experimental results

As shown in the above three result graphs (executable files related to the article's open source), Number of faces (N of F), Number of points (N of P), origin exe (O), my reproduce code (M).

The comparison shows that my reproduced code is as effective as the open source exe. For the calculation of large model, the calculation time is shorter.

6 Conclusion and future work

1.This article still depends on the plane detection algorithm to a certain extent, which is a major limitation in the field of structural reconstruction of buildings. In the future, we can focus on reducing the dependence on plane detection.

2.In this paper, the sculpting method is used to obtain the model surface, avoiding the use of optimization algorithm to extract the surface. This idea may be extended in a more mathematical way.