Optimization of Texture Rendering of 3D Building Model Based on Vertex Importance

Wenfei Shen[1], Liang Huo[2] , Tao Shen[3] *, Miao Zhang[4], Yucai Li[5]

[1] Beijing University of Civil Engineering and Architecture, Beijing, China - swf6661@outlook.com
[2] Beijing University of Civil Engineering and Architecture,Beijing, China - huoliang@bucea.edu.cn
[3] Beijing University of Civil Engineering and Architecture,Beijing, China - shentao@bucea.edu.cn
[4] Beijing University of Civil Engineering and Architecture, Beijing, China - ZM000419@163.com
[5] Beijing University of Civil Engineering and Architecture, Beijing, China - liyucai1211@163.com

Commission II/WG II/3 - 3D Scene Reconstruction for Modeling & Mapping

KEY WORDS: 3D building model, Vertex importance, 2D space boxing algorithm, Texture merging, Remapping

ABSTRACT:

In 3D building models, a large number of texture maps with different sizes increase the number of model data loading and drawing batches, which greatly reduces the drawing efficiency of the model. Therefore, this paper proposes a texture set mapping method based on vertex importance. Firstly, based on the 2D space boxing algorithm, the texture maps are merged and a series of Mipmap texture maps are generated, and then the vertex curvature, texture variability and location information of each vertex are calculated, normalized, and weighted to get the importance of each vertex, and then finally, different Mipmap-level textures are remapped according to the importance of the vertices. The experiment proves that the algorithm in this paper can reduce the amount of texture data on the one hand, and avoid the rendering pressure brought by the still large amount of data after merging on the other hand, so as to improve the rendering efficiency of the model.

1. INTRODUCTION

Currently, the rendering optimization of texture data mainly includes texture compression and texture merging, and the existing texture merging tools rely on manual experience, with a low degree of automation, and a large limitation of application (Liu Tianyi, 2023). For example, NVIDIA's texture tools, Autodesk's 3DMax software's UVW-UnWrap tool, and so on. The above texture merging tools more or less need manual intervention and experience in using them, and cannot realize the whole automation, which is obviously not suitable for the batch processing of texture data of large-scale 3D city models. Xuefeng Dai (2015) proposed an automatic multi-texture merging method based on greedy and annealing algorithms; Qing Zhu (2021) proposed a two-dimensional spatial crating algorithm to optimize the redundant textures during texture merging; Bao X(2020) proposed a method to simplify the texture data and compression management using fractal quadtree, and used fractal texture compression to create a multi-resolution texture data structure with a quadtree structure, so that the 3D models of buildings to achieve dynamic visualization at different scales; Zhang (2021) introduced a size-adaptive texture atlas method without loss of accuracy and addition of extra storage space; Jae-Ho N (2023) An improved ETC2 coding technique for real-time, high-quality texture compression ; Guillaume L (2019) provided some methods to evaluate the quality of texture compression. Among them, texture compression algorithms are complex and degrade texture quality, and for models in large scenes with merged textures, the amount of data is still large. In computer vision, Mipmap texture is widely used, Wang Zhenni (2019) maintains its fast map generation speed advantage based on seamless Mipmap filtering without compromising the map generation speed advantage; Sungkil L (2009) nonlinear interpolation of mipmap images generated from pinhole images to achieve depth of field effect.

In the process of texture mapping, texture offset and misalignment often exist on the model surface, Cheng Yan(2023) constrains the positions generated by the texture seams to mitigate the misalignment on both sides of the seams, and Yongkai Y(2019) introduces coded markers as control points to assist texture mapping in order to avoid the dependence on the texture features and geometric model. Mipmap texture is a texture image containing a series of texture images, each of which is a low-resolution representation of the previous one, with the height and width of each level of the image being half the size of the previous one. This paper draws on this idea and proposes a method for mapping different levels of Mipmap textures based on vertex importance, in order to address the problem of rendering pressure caused by the still large amount of data after merging. Firstly, the texture maps are merged into a texture atlas based on the 2D space boxing algorithm, and then vertex curvature, texture variability and vertex distance from the model centroid are computed, normalized, and weighted to obtain vertex importance. In computer vision, textures with different Mipmap levels are dynamically displayed according to the viewing angle distance to improve the rendering efficiency. In this paper, different levels of Mipmap textures are mapped based on vertex importance to improve the rendering speed.

The remainder of this paper is organized as follows. In Section 2 , a texture map merging method based on the 2D space boxing algorithm is described in detail, and different levels of Mipmap textures are mapped by calculating the importance of vertices. The experimental results are presented in Section 3 are presented and discussed and analyzed. Finally, the experimental results are presented and discussed and analyzed in Section 4 in which the paper is briefly summarized.

2. METHOD

Figure 1 shows the process of merging texture maps into texture atlases and generating Mipmap textures, as well as the process of remapping Mipmap textures onto a mesh based on vertex importance. The method in this paper can be roughly divided into the following four steps:

1. All texture maps are merged into a texture atlas according to the 2D space boxing algorithm;
2. Generate a series of Mipmap textures, discarding the less visually appealing Mipmap textures;
3. Vertex curvature, texture variability, and vertex distance are calculated, normalized, and weighted to obtain vertex importance;
4. Group all vertices and remap the texture maps of each group onto the original mesh based on vertex importance.

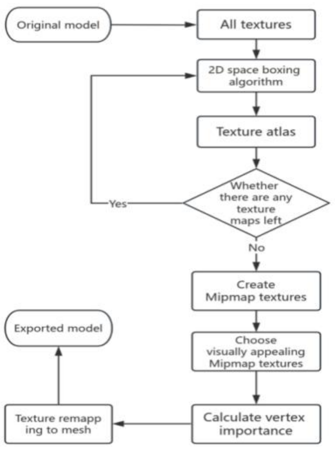
Original model
All textures
2D space boxing algorithm
Texture atlas
Whether there are any texture maps left
Yes
No
Create Mipmap textures
Choose visually appealing Mipmap textures
Calculate vertex importance
Texture remapping to mesh
Exported model

Figure 1.Flowchart of the method of this paper

2.1 Merge Texture and Generate Mipmap Texture

In the rendering process of 3D building models, a large number of texture maps with different sizes will lead to more loading batches, which will undoubtedly reduce the rendering efficiency (Liu Zhendong, 2022). Therefore, in this paper, we consider merging many texture maps into a texture map set to reduce the loading batches and also reduce the data volume. As shown in Figure 2, based on the 2D space boxing algorithm, many textures can be merged into one texture atlas, and in the boxing process based on the greedy algorithm to fill the texture continuously, and at the same time, the texture area and height are considered, the result is more reasonable, and the data volume is smaller.

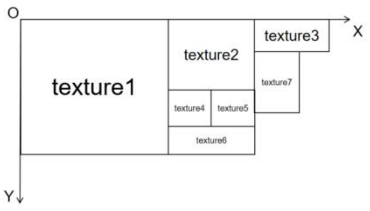

Figure 2. Schematic diagram of texture merging

The specific algorithm flow is as follows:

1. Sort all texture maps according to height from largest to smallest and define the set of texture maps as R={W, H};

$$W = \min(R_{\max}, \frac{\sum_{i=0}^{n} S_i}{h_{\max}}) \quad (1)$$

If division is not possible, round up while ensuring that the result is an integer power of 2;

$$H = \min(R_{\max}, h_{\max}) \quad (2)$$

where S_i = the area of the texture map

$h_{\max}$ = the height of the first texture map

$R_{\max}$ = the maximum resolution supported by the computer.

2. Fill from left to right according to height size, if the height is the same then sort by area from largest to smallest;
3. When the horizontal direction is filled or the remaining space can no longer be filled, fill in the vertical direction in turn. Loop through the remaining texture maps to find a texture map that exactly matches the width of the remaining space in the vertical axis direction, or if none exists, fill them in order of height and area size until the set of texture maps is filled completely or can no longer be filled;
4. If there are remaining texture maps, create another texture map set and repeat steps 1, 2, and 3.

After the texture merging is completed, the texture coordinates of each vertex will also be changed, and the new texture coordinates corresponding to each vertex in the texture atlas need to be recalculated (Wang, 2021). The new texture coordinates can be calculated by the following formula:

$$U = a * u + b \quad (3)$$

$$V = c * v + d \quad (4)$$

Where u, v = the original texture coordinates

a = width of the original texture/width of the merged texture

c = height of the original texture/height of the merged texture

b, d = horizontal coordinates, vertical coordinates of the lower left corner of the merged texture in the coordinate system of the original texture

As shown in Figure 3, for all the texture mapsets after merging, a corresponding series of Mipmap textures are generated, and the Mipmap textures with poor visual effects are discarded, which on the one hand avoids the visual effects from being affected too much, and on the other hand also reduces the texture map loading batches and improves the rendering efficiency.

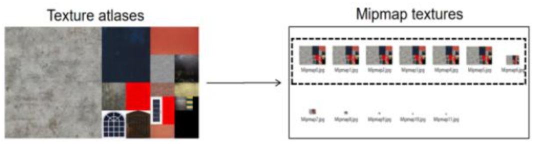

Figure 3. Mipmap texture map

2.2 Calculating Vertex Importance for 3D Building Models

Since each part of a 3D building model receives different attention, this paper chooses to calculate the importance of each vertex of a 3D building model from three aspects (Barone, 2020), and maps textures with different resolutions according to the vertex importance.

In this paper, vertex importance is obtained by calculating vertex curvature, texture variability and vertex position, normalized and weighted. Vertex curvature can quantitatively represent the degree of concavity and convexity of vertices, where curvature refers to approximate curvature, and since the surface of the model is a second-order nondifferentiable surface, the surface of the model can be regarded as approximately smooth (Xiang, 2022), and places with obvious degree of concavity and convexity are more likely to attract attention. Texture variability is mainly reflected by color difference, and it is easier to attract attention when a color contrasts significantly with the surrounding colors (Li Guoli,2019). Vertex position indicates the distance from the vertex to the center of the model, and vertices that are closer to the center are more likely to attract attention.

Vertex curvature can reflect the degree of concavity and convexity of the detailed features in the feature region of the model. In the flat region of 3D model, the curvature is small; in the 3D model crease and inflection point feature region, the curvature is large (Li Shaoqing, 2021). As shown in Figure 4, the vertex curvature calculation in this paper combines the triangle area, and first calculates the vertex v_i unit normal vector n_{vi} :

$$n_{vi} = \frac{\sum_{i=1}^{k} S_i n_i}{\left\| \sum_{i=1}^{k} S_i n_i \right\|} \tag{5}$$

where n_i = the unit normal vector of each triangle facet in the vertex one ring field

S_i = the area of each triangle.

The curvature (cur) of this vertex can then be calculated: the

$$\text{cur} = \frac{\sum_{i=1}^{k} \alpha(n_{vi}, n_i)}{k} \tag{6}$$

where $\alpha(n_{vi}, n_i)$ = the angle between the vertex normal vector and the unit normal vector of each neighboring triangle facet

k = the number of triangles in the one-loop domain

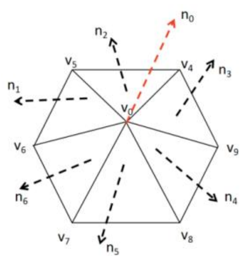

Figure 4. Schematic diagram of the normal vector of a triangle

Texture discreteness is the color difference between a vertex and its neighboring vertices. A color is also more noticeable when it contrasts significantly with the surrounding colors (Li P, 2021). Before calculating the texture discreteness of a vertex, it is necessary to find the corresponding pixel point on the texture atlas based on the texture coordinates of the vertex.

$$\text{Pixel\_x} = U * (\text{texWidth} - 1) + 0.5 \tag{7}$$

$$\text{Pixel\_y} = V * (\text{texHeight} - 1) + 0.5 \tag{8}$$

where U,V = the new texture coordinates after merging

texWidth, texHeight = the size of the merged texture atlas

In HSV color space, hue (hue) takes values in the range [0, 360) degrees. Since hue is a cyclic value, its periodicity is also considered. The hue calculation formula is as follows (Zohra H, 2015):

$$\text{hue} = \begin{cases} \theta (G \geq B) \\ 2\pi - \theta (G < B) \end{cases} \tag{9}$$

where $\theta = \cos^{-1}\left[\frac{[(R-G)+(R-B)]/2}{\sqrt{(R-G)^2 + (R-B)(G-B)}}\right]$,

R, G, and B = the RGB values of the pixels in the texture map.

The color difference (cod) between the vertex and other vertices in a ring field is then calculated based on the hue:

$$\text{cod} = \frac{\sum_{i=0}^{n} (\| |hue| - |hue_i| \|)}{n} \tag{10}$$

Finally, the closer the vertices are to the center of the model, the easier it is to draw attention to them throughout the 3D building model. The coordinates (X,Y,Z) of the vertex at the center of the model can be defined as:

$$\begin{cases} X = \frac{\sum_{i=0}^{n} X_i}{n} \\ Y = \frac{\sum_{i=0}^{n} Y_i}{n} \\ Z = \frac{\sum_{i=0}^{n} Z_i}{n} \end{cases} \tag{11}$$

Then calculate the distance (dis) from each vertex to the center.

$$\text{dis} = \sqrt{(X-x)^2 + (Y-y)^2 + (Z-z)^2} \tag{12}$$

In order to eliminate the unit differences between different features, the normalization process can transform these three different units of data into relatively consistent ranges, as exemplified by the normalization of vertex curvature:

$$\mathrm{cur}_{norm} = \frac{cur - cur_{\min}}{cur_{\max} - cur_{\min}} \tag{13}$$

In order to avoid the disadvantages of subjective assignment, this paper uses the entropy weight method (Li Zhanshan, 2022) to calculate the weights of each indicator after normalization with the following steps:

1. There are three indicators in this paper, which form the original data matrix below:

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & x_{n3} \end{pmatrix} \quad (x_{ij},\ i=1,2\ldots n;\ j=1,2,3) \tag{14}$$

2. Since the units of measurement of the indicators are not uniform, they need to be standardized before they can be used to calculate the composite indicators, thus solving the problem of homogenization of the values of the different qualitative indicators. In addition, positive and negative indicators have different meanings (the higher the value of a positive indicator, the better, and the lower the value of a negative indicator, the better), so different algorithms are needed to standardize the data for positive and negative indicators:

$$y_{ij} = \begin{cases} \dfrac{xi_j - \min(X_j)}{\max(X_j) - \min(X_j)} & \text{Positive indicator} \\ \dfrac{\max(X_j) - x_{ij}}{\max(X_j) - \min(X_j)} & \text{Negative Indicator} \end{cases} \tag{15}$$

3. Calculate the weight of the indicator P.

$$p_{ij} = \frac{y_{ij}}{\sum_i^n y_{ij}} \quad i=1,2,3\ldots.n;\ j=1,2,3 \tag{16}$$

4. Calculate the information entropy E.

$$E_j = -\frac{1}{\ln(n)} \sum_{i=1}^{n} p_{ij} \ln(p_{ij}) \tag{17}$$

5. Determine the weights of the indicators W.

$$W_j = \frac{1 - E_j}{\sum (1 - E_j)} \quad (j=1,2,3) \tag{18}$$

The importance of each vertex is then computed by weighting the vertex curvature, texture variability and vertex position (Iov):.

$$\mathrm{Iov} = W_1 * cur_{norm} + W_2 * \mathrm{cod}_{norm} + W_3 * \mathrm{dis}_{norm} \tag{19}$$

2.3 Remapping

Remap the merged texture atlas with the Mipmap texture onto the mesh based on vertex importance as follows:

1. Based on the mapping relationship between the texture coordinates corresponding to each texture atlas and the coordinates of each vertex, all the vertices are divided into large groups, with the number of groups = the number of texture atlases;
2. The vertices of each large group are sorted in descending order of vertex importance, and then divided equally into groups, with the number of groups = the number of Mipmap textures retained in each texture atlas + 1;
3. The texture corresponding to each group is remapped onto the mesh according to the vertex importance, and the higher the value of the vertex importance, the higher resolution texture map is mapped for that vertex, as shown in Figure 5.

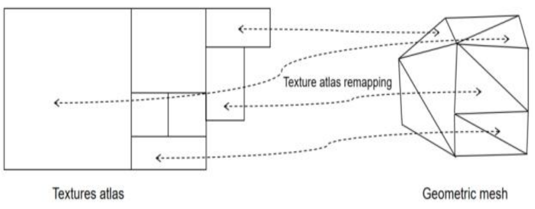

Figure 5. Schematic diagram of texture remapping

3. RESULTS AND ANALYSIS

In this paper, python is used as the development language, and three different building models (Figure 6) are selected for experiments to avoid the chance of experimental results. The experimental data in this paper is text type data in obj format, which is easier to edit than other binary data.

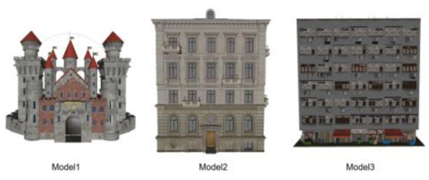

Figure 6. 3D building models selected for this paper

The experiments will analyze the results from four aspects: texture data volume, loading time, frame rate and visualization effect, and compare the method of this paper with the traditional method of texture optimization, and verify the superiority of this paper's algorithm through analysis.

In terms of the amount of texture data, it can be seen from Table 1 that only merging or compressing textures can reduce the amount of texture data by more. And in this paper, all texture maps are firstly merged into one texture atlas based on the 2D boxing algorithm, and then a series of Mipmap texture maps are generated based on the texture atlas, although this process will increase the extra storage space, the overall amount of texture data is reduced due to the merging process of the original texture maps.

		Texture data volume/MB			
		Original model	Texture Gallery	Mipmap Texture	Total data volume
	Method of this paper	2.07	1.46	0.46	1.92
Model 1	Merge	2.07	1.46	0	1.46
	Compression	2.07	0	0	0.98
	Method of this paper	91	23.2	7.19	30.39
Model 2	Merge	91	23.2	0	23.2
	Compression	91	0	0	20.3
	Method of this paper	63.8	26.3	8.15	34.45
Model 3	Merge	63.8	26.3	0	26.3
	Compression	63.8	0	0	20.3

Table 1.Comparison of texture data volume

In terms of loading and rendering efficiency, it can be seen from Table 2 and Figure 7 and Figure 8 that although compression can reduce the amount of more texture data, the number of texture files remains unchanged, resulting in an insignificant improvement in the model loading time and rendering efficiency; whereas, the number of texture files after merging is sharply reduced to one, which obviously reduces the texture map loading batch, and can improve the rendering efficiency dramatically; whereas, this paper's method combines the Mipmap texture, although the number of files is relatively increased, but in the process of model rendering, the

Depending on the vertex importance a texture map with a lower resolution can be selected and rendered more efficiently.

		Number of documents/p.	Load time/s	Rendering frame rate/fps
	Original model	16	6.22	28
Model 1	Merge	1	3.42	36.2
	Compression	16	4.55	33.5
	Method of this paper	5	2.78	47.6
	Original model	40	10.18	36
Model 2	Merge	1	3.95	44.8
	Compression	40	8.29	40.2
	Method of this paper	5	3.41	51.9
	Original model	10	3.42	78.1
Model 3	Merge	1	2.51	85
	Compression	10	2.67	81.9
	Method of this paper	5	2.22	95

Table 2.Comparison of loading and rendering efficiency of different models

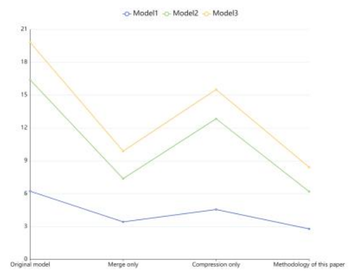

Figure 7. Loading time for different models

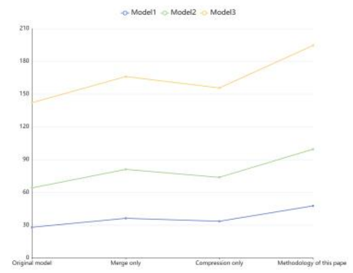

Figure 8. Rendering frame rate for different models

In terms of visualization effect, the overall visualization effect of the three methods is almost the same, but in terms of specific details, as can be seen from Figure 9, the rendering effect of this paper's method is slightly worse than that of the merging-only method in the center position of Model1, but the effect of this paper's method is much better in terms of compression effect. In Model2, in the part of the building façade where the vertices are protruding, the rendering effect of this paper is the same as that of the merge-only method because the effect of the curvature of the vertices is considered in this paper and the region is very close to the center of the model. In Model3, for the places with large texture differences, the method in this paper has only a slight blurring effect compared to the merge-only method, and performs better for the compression effect. Although in some details, the rendering effect of this paper is a little bit worse than that of merge-only, but because this paper maps different levels of Mipmap textures based on the vertex importance, and maps high-resolution textures at places with high vertex importance and low-resolution textures at places with low vertex importance, the rendering speed of this paper's method is faster.

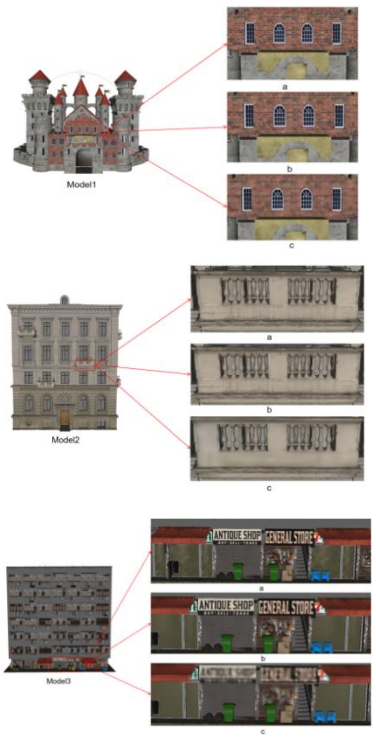

Figure 9. Comparison of rendering results (a: merge only, b: this paper's method, c: compression only)

Summarizing the above analysis, the method in this paper merges the texture maps based on the 2D crate algorithm, and even though the generation of Mipmap textures requires extra storage space, the overall texture data volume is still reduced, and the texture loading batch is also reduced, which improves the rendering efficiency. Mapping different levels of Mipmap textures according to the vertex importance still renders well in the parts of the model that are under attention. Therefore, the method in this paper has a better overall performance in terms of both rendering effect and data volume compared to the other two texture optimization methods.

4. CONCLUSION

This paper proposes a vertex importance-based texture set mapping method, and experiments show that, compared with traditional texture optimization methods, this paper's method reduces the amount of texture data on the one hand, and on the other hand, mapping texture sets of different resolutions based on vertex importance still maintains good rendering effects in the parts of 3D building models that are under attention, which solves the difficult problem that can't be achieved both in terms of visualization effect and rendering efficiency, and This is of great significance for improving the rendering efficiency of large-scale urban 3D scenes. Therefore, we will continue to study how to further apply the method of this paper to large-scale urban scenes, as well as study the simplification of mesh data to further reduce the data volume of the model.

REFERENCES

Bao X ,Zhou Q G ,Yue T , et al.RESEARCH ON 3D BUILDING VISUALIZATION BASED ON TEXTURE SIMPLIFICATION AND FRACTAL COMPRESSION[J].ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences,2020,XLII-3/W10179-185.

Barone A V S C ,Nunes D G ,Felipe L S S D , et al. Importance of Vertices in Complex Networks Applied to Texture Analysis.[J].IEEE transactions on cybernetics,2020,50(2):777-786.

Cheng Y., Ge Liang, Zhang Fan et al. Automatic texture reconstruction of building models by inclined photography[J]. Journal of Surveying and Mapping,2023,52(09):1528-1537.

L. Li, L.P. Shao, P.P. Ren. Generative information hiding with differential clustering and error texture synthesis[J]. Chinese Journal of Image Graphics,2019,24(12):2126-2148.

Guillaume L ,Michael L ,Adrien P , et al. A Psy-chophysical Evaluation of Texture Compression M-asking Effects.[J].IEEE transactions on visualization and computer graphics,2019,25(2):1336-1346.

Jae-Ho N .QuickETC2-HQ: Improved ETC2 encoding techniques for real-time, high-quality texture compression[J].Computers Graphics,2023,116308 -316.

Li P, et al. A simplification algorithm for real-world 3D models by fusing texture information. Surveying and Mapping Science 46.10(2021):151-158+.166. doi:10.16251/j.cnki.1009-2307.2021.10.020.

Li Shaoqing, et al. A Simplified Algorithm for Edge Folding of 3D Building Models Taking Into Account Angular Errors. journal of Wuhan University (Information Science Edition) 46.08(2021):1209-1215.doi:10.13203/j.whugis201902 69.

Liu Tianyi, Gan Linlu. Multi-texture 3D model simplification algorithm based on vertex clustering[J]. Mapping and Spatial Geographic Information,2023,46(07):221-224.

Liu Zhendong, et al. A Model Simplification Algorithm for 3D Reconstruction.Remote Sensing 14.17(2022):4216-4216.

Li Zhanshan,Yang Yunkai,Zhang Jiachen. A filtered feature selection algorithm based on entropy weight method[J]. Journal of Northeastern University(Natural Science Edition),2022,43(07):921-929.

Sungkil L ,Jounghyun G K ,Seungmoon C .Real-time depth-of-field rendering using anisotropically filtered mipmap interpolation.[J].IEEE transactions on visualization and computer graphics,2009,15(3):453-64.

Wang Biao, Wu Guoping, Zhao Qiang, Li Yaozhu, Gao Yiyuan, She Jiangfeng. A Topology-Preserving Simplification Method for 3D Building Models[J]. ISPRS International Journal of Geo-Information,2021,10(6).

Xiang Hanyu, Huang Xianfeng, Lan Feng, Yang Chong, Gao Yunlong, Wu Wenyu, Zhang Fan. A Shape-Preserving Simplification Method for Urban Building Models[J]. ISPRS International Journal of Geo-Information,2022,11(11).

Xuefeng Dai, Jianghan Xiong, Jianya Gong. An automatic multi-texture merging method for 3D city model[J]. Journal of Wuhan University (Information Science Edition), 2015, 40(3): 347-352.

Yongkai Y ,Hailong C ,Xiangfeng M , et al.Texture mapping based on photogrammetric reconstruction of the coded markers.[J].Applied optics,2019,. 58(5):A48-A54.

Zhang, X.; Liu, W.; Liu, B.; Zhao, X.; Hu, Z. Size-Adaptive Texture Atlas Generation and Rema-pping for 3D Urban Building Models. ISPRS Int.J.Geo-Inf. . 2021, 10, 798. https://doi.org/10.3390/ijgi10120798.

Zhenni Wang, et al. Seamless Mipmap Filtering for Dual Paraboloid Maps.Computer Graphics Forum 38.7(2019):437-448.

Zhu, Q., et al. A 2D crate method for fine building fragmented texture optimization. Journal of Southwest Jiaotong University 56.02(2021):306-313.

Zohra H ,Kamal H ,Gérard J P .A fast algorithm for texture feature extraction from gray level aura matrices[J]. Systems and Signal Processing,2015,954-61.