Object based manipulation with 3D scenes in mobile environments

Ladislav Cmolik[1], Zdenek Mikovec[1], Pavel Slavik[1]

[1] Czech Technical University in Prague
Karlovo namesti 13, Prague 2, Czech Republic
`{cmolikl, xmikovec, slavik}@fel.cvut.cz`

Abstract. This paper deals with the problem of visualization of 3D scenes and navigation in them on mobile devices. A visualization technique that enables visualization and annotation of objects in a 3D scene on mobile devices is presented. This technique is based on transformation of the 3D scene to a 2.5D representation in 2D vector graphic. The input of the 3D scene is given in the VRML format and the output 2D vector graphic format is SVG.

1   Introduction

In the mobile environment, where the mobile devices have low computational power and small screens, it is very difficult to efficiently visualize 3D scenes. Especially the problem of navigation in the 3D scene is noticeable. The navigation in the 3D scene in general requires concurrent usage of special keys, to switch between navigation modes (move, pan, rotate …), in combination with pointing device (mouse, joystick, etc.). Mobile devices do not enable such navigation comfortably. Moreover, common users are not trained to work in environment with so many degrees of freedom combination of which results in the lost of orientation in the 3D scene. As mentioned in [7] the users prefer a 2D graphic environment or a combination of the 2D and the 3D environment.

1.1   State of the Art

In this section we present existing solutions to the visualization of 3D scenes on mobile devices. The solutions can be divided into three main categories.

In the first category are *server-side methods*. These methods [4] provide remote rendering of the 3D model as a reaction on user-interaction-requests from the mobile device and sends rendered raster images back to the mobile device.

In the second category are *client-side methods*. These methods [9] render the 3D scene fully on the mobile device.

In the third category are *hybrid-side methods*. These methods [3], [2] balance the work load between the server and the client to decrease network communication and improve performance on the client side.

The server-side and the client-side methods do not give satisfactory results. The server-side methods achieve frame-rate around 7 fps [4], [9] due to time spent on rendering on the server and the transfer through wireless network. The client-side methods achieve approximately the same frame-rate as server-side methods, but due to low computational power of mobile devices. With so low frame-rate and so small resolution of raster images the system is suitable only for visualization of one 3D model at time. The visualization of a larger 3D scene will result in hard navigation in the scene and in lost of orientation of the user.

The hybrid-side methods achieve suitable frame-rate (20 fps), but allow only low visual quality of the rendered scene due to line oriented visualization.

Notice that none of the methods focuses on the navigation problem or allows user to interact with objects in the scene. Our approach described in the next section addresses also these problems.

1.2 Our Approach

Our approach is a hybrid-side method. It is based on transformation of the 3D scene by means of projection to the 2D vector image. The transformation preserves the object oriented representation of the 3D scene - each 3D object is rendered individually. The resulting 2D vector image has 2.5D representation (each 2D object is located in its own projection plane and the objects are ordered by distance from camera). This object oriented approach allows the user to individually manipulate the objects in the 2.5D scene. To implement these features the vector graphic format SVG (Scalable Vector Graphics) [5] was chosen as the 2.5D scene description format. SVG is object oriented and supports zooming, panning and interaction with the objects. The whole transformation and visualization process is distributed between a server and a mobile device. The transformation is performed on the server side. The 2.5D representation of the 3D scene is sent to the mobile device, where it is visualized. (see Figure 1). The visualization of 2D data is computationally less expensive than visualization of 3D data and therefore the response of the mobile device on user interactions is faster.
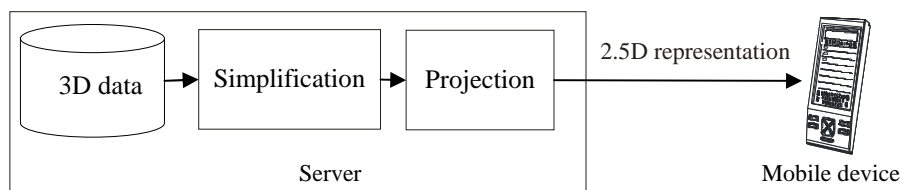
Figure 1. Distribution of the visualization process of 3D scene

The following text describes the use case where the visualization process described above will be used.

1.3 Use Case

Let us consider a person equipped with a PDA visiting a part of a city (in general a 3D scene). His/her task is to investigate the spatial structure of this city part. So s/he retrieves the information needed from the knowledge management system through the wireless network. The data delivered to the client are not in the original 3D form (that is not suitable for the usage in mobile environment) but they have been transformed to 2.5D form by means of process described above. The position of the user (avatar) in the 3D scene will be used as a viewpoint position for the transformation to the 2.5D scene representation.

The person is able to locate objects in his/her region of interest (e.g. s/he might be interested in placement of some objects in the scene). These objects might be in general hidden behind other objects in 2.5D scene from his current point of view. To perform these activities the person needs specific tools at his/her disposal to manipulate, annotate, interact and change visualization modes of the objects on PDA. These tools should be more flexible in comparison with tools available in traditional 3D browsers. Moreover they should be less demanding from the point of computational power (as PDA is much less powerful than standard PCs). In our case such a tool is an SVG viewer.

2 Implementation of the Transformation Process

Our approach is based on a distributed visualization pipeline (server/client), see the Figure 2. The input of the visualization pipeline is a scene graph obtained by the 3D scene parser. The visualization pipeline can be described as follows:
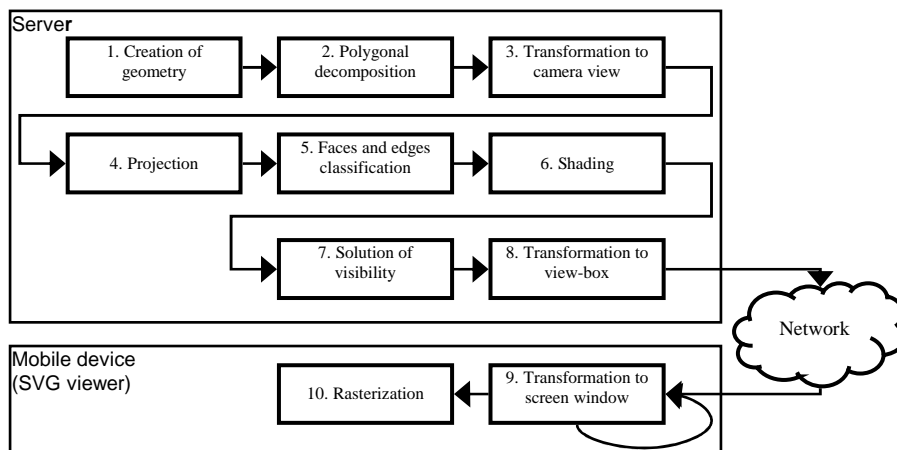

Figure 2. The visualization pipeline

1. The geometrical representation is created for each 3D object in the 3D scene.
2. All polynomial faces (Bezier patches) are decomposed to convex polygons.
3. The 3D scene is transformed to the to the camera view.
4. The 3D scene is projected to the viewing plane.
5. All edges are classified to front, border and hidden ones. All faces are classified to front and back.
6. All faces are shaded.
7. The visibility in the 3D scene is solved.
8. The viewing plane is transformed to the view-box.
9. The view-box is transformed to the screen window.
10. The objects in screen window are rasterized on the screen.

Steps 1 to 8 are performed on the server side by our system and steps 9 and 10 are performed on the mobile device by an SVG editor. A mobile SVG editor has been implemented in the framework of the Mummy project, for more details see [6]. Note that all operations on the server side have to be performed in the object space (individual 3D object transformation), while the rasterization step (10) is performed on the mobile device.

The structure of the visualization pipeline is a standard one, except the back-faces are not culled and the solution of visibility is performed in object space. All algorithms are modified to be able to handle the back-faces. Our innovative approach to the solution of visibility in object space is described in the next section.

2.1 Solution of Visibility

The rasterization step is performed on the mobile device in the last step of the visualization pipeline, therefore the visibility of a 3D scene, which is solved on the server side, has to be solved in the object space and none of the raster oriented methods (e.g. z-buffer, ray-tracing) is suitable for our purposes.

The most appropriate method meeting most of our criteria is the painter's algorithm. In our approach the painter's algorithm generates an ordered list of 2D projections of 3D objects. Each individual 2D projection is in vector representation (SVG). The ordering in the list is generated according to the distance of the object from the camera view (viewpoint). The painter's algorithm has following properties:

It is face oriented and allows easy description of the flat shaded faces in the 2.5D scene.
It supports various visualizations of the 2.5D scene, because it contains information about all faces, edges and vertices in original 3D scene.
It is capable to handle intersecting faces.
It is balanced between the rendering speed and the size of the final 2.5D scene.

The painter's algorithm preserves almost the same number of faces in the output 2.5D scene (including the backfaces) as in the input 3D scene. The information about the ordering of the 2D objects in the 2.5D representation is at disposal. The fact that

the 2.5D scene contains information about all faces, edges and vertices is very important for visualization and annotation of the scene. It is described in the next section.

Our implementation of the painter's algorithm is the full painter's algorithm. The painter's problem and an occurrence of intersecting faces are handled correctly. The painter's algorithm, as well as our implementation, has the $O(n^2)$ computational complexity - n is the number of faces in the 3D scene. More information about our implementation of the painter's algorithm can be found in [1].

3 Various Visualization Modes of Projected Data

The painter's algorithm preserves the information about all objects and also about their order with respect to the camera view. Investigation of the 3D scene in 2D space does not allow the user to access all the 3D information (like information about the objects hidden around the corner). Our approach reduces this problem benefiting form the 2.5D representation and by introducing special modes of objects visualization.

The SVG used for 2.5D representation of the 3D scene is an XML based format, therefore the CSS (Cascading Style Sheets) can be used to define various visualization modes of the 2D objects. To each SVG file a CSS representing the visualization modes is attached. Moreover SVG enables interactive and selective change of a visualization mode of an object in the SVG file. This support of various visualizations compensates in some way the movement of avatar in the 3D scene. We will demonstrate this feature on following scenario.

Let us consider that the user equipped with PDA is surveying a street with two rows of houses and an object (head of monkey). The corresponding visualization of the original 3D scene can be seen in Figure 3. The task of the user is to explore the scene.
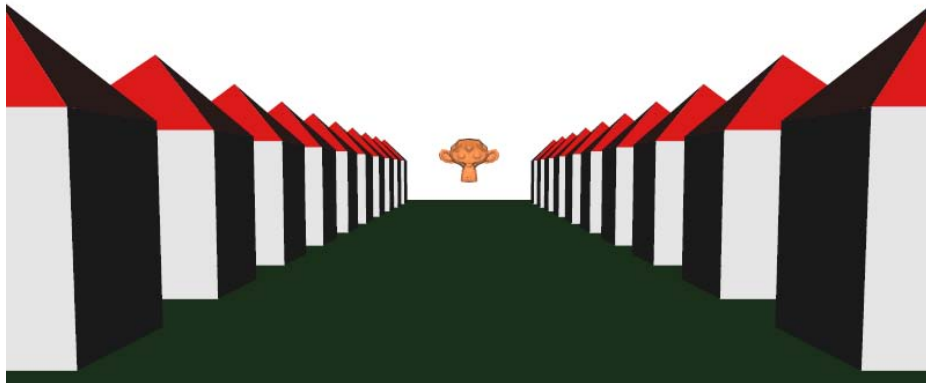
Figure 3. The initial visualization of the 3D scene (a street with two rows of houses)

The user can view the 2.5D scene in various visualization modes (the objects can be displayed in solid, semitransparent or wire-frame mode). Moreover zooming,

panning can be used without sending any request to the server. The user can also annotate the objects in 2.5D scene.
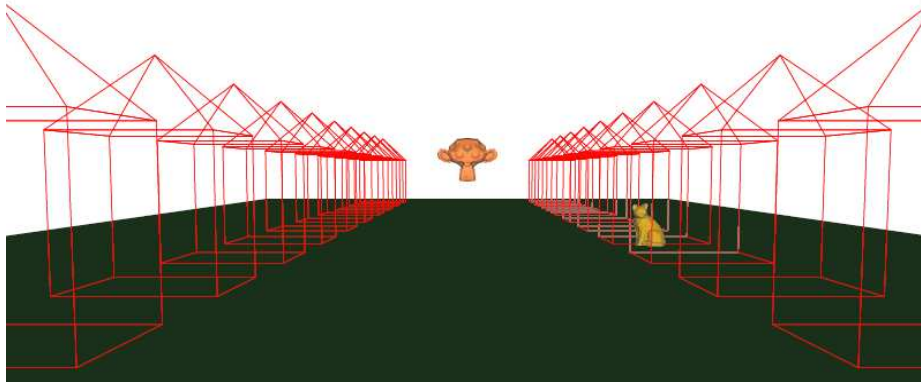
**Figure 4**. The houses visualized in wire-frame mode. A cat hidden behind houses could be identified

The user needs to examine the scene in detail. The 2.5D scene in Figure 3 is not informative enough, because some objects can be hidden behind houses. The user changes the visualization mode of all houses to wire-frame mode to see if some objects are hidden behind houses (see Figure 4).

Now the user found out that a cat is hidden behind houses. The user switches all houses back to solid visualization mode, except houses before the cat (see Figure 5).
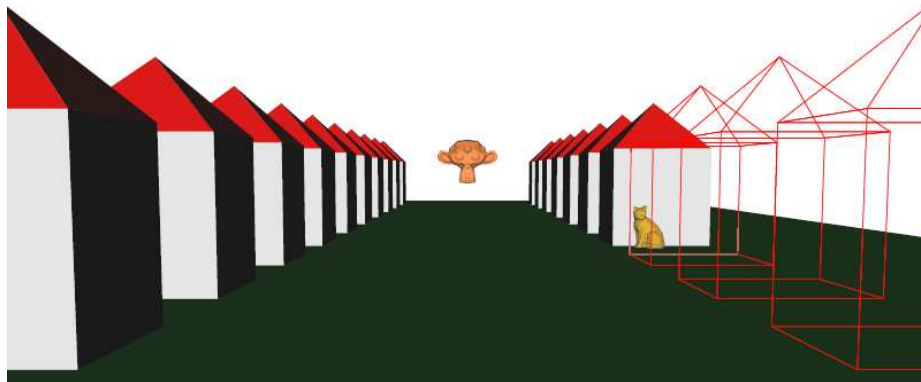
Figure 5. The houses are visualized in solid mode, except the houses before the cat.

The user wants to attach an annotation to the cat. For annotation purposes he zooms the cat in order to create unambiguous relation between the annotation and the cat (see Figure 6).
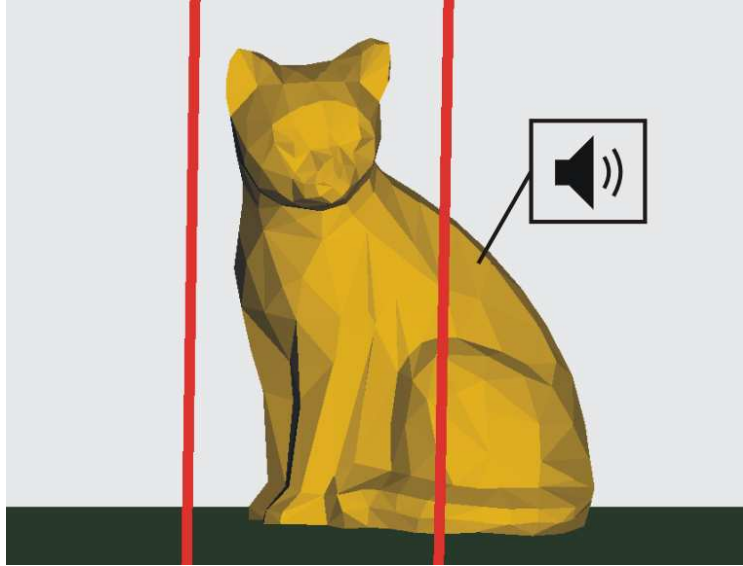
Figure 6. The user zoomed to the cat in order to create unambiguous relation with the annotation

Now the user can create a multimedia annotation, e.g. voice record and attach it to the cat.

Notice that the whole process of scene exploration is done only on the client side (no requests to server are sent). The request is sent only in the phase of annotation creation.

4 Testing

The testing was focused on the performance of our application. The performance test was focused on the painter's algorithm, because the painter's algorithm consumes a significant amount of time from the whole time spent on the visualization. We have created a special testing object with high complexity (large number of faces). Several 3D models of monkey with different level of detail were generated. The models have been visualized by our application and the time spent on the painter's algorithm has been measured. The results are as follows.

The size of SVG file depends linearly on the number of faces. The computational complexity depends quadratically on the number of faces. This corresponds with the $O(n^2)$ computational complexity of the painter's algorithm. The measurement showed that the performance of the current implementation of the painter's algorithm is sufficient for scenes with relatively small number of faces (up to 2000). In such a case the time of 2.5D scene generation requires about 2 seconds. The typical scene used in our use case contains up to 1000 faces.

We should also mention that the size of the SVG file is always larger than the size of the corresponding VRML file. The SVG file contains much more detail description of the scene, e.g. the color is stored in the SVG file for every face. In VRML is usually defined one color for whole object. The size of the file does not create any problems during data transmission as it can be compressed with high compression ratio.

5 Conclusion

In this work a new visualization method of 3D scenes suitable for mobile environment has been presented. The visualization process is distributed between the server and the mobile device (client). The 3D scene is projected to 2.5D representation on the server side and this 2.5D representation is delivered through the network to the mobile device. The users can interactively change visualization modes of objects in the 2.5D scene (e.g. solid, semitransparent or wire-frame mode). The users can also zoom and pan the 2.5D scene. Moreover they can annotate the objects in the 2.5D scene. The possibility to choose various visualization modes of objects in the 2.5D scene allows the user to investigate the information normally available only in 3D representation of the scene (e.g. objects hidden around the corner).

The most important advantage of our approach described is easy interaction with the 3D scene in 2.5D representation while at the same time the possibility to investigate the structure of the 3D scene is preserved.

The functional prototype of the application was implemented, tested on various 3D scenes and the performance was measured. The measuring has proven that the performance of the prototype is satisfying for 3D scenes containing up to 2000 faces (for common mobile devices).


5.1 Future Work

The first part of the future work will be concentrated on the speeding up the painter's algorithm in order to cover other applications. Improvement will be achieved by usage of interval trees in the painter's algorithm to enable faster testing of bounding-box intersections.

The second part of the future work will be focused on user testing. The user testing should prove that users are able to explore 3D scenes in our 2.5D representation and achieve the more or less the same amount of information as when exploring the 3D scene, but in form more suitable for mobile devices and mobile environment.

The third part of the future work will be focused on utilization of the semantic description of the 3D scenes in the filtration process. The semantic description stored in the OWL [8] (Web Ontology Language) will be used for selective changes of the visualization modes of the objects in the 2D scene. The semantic description is usually application oriented - this means that the filtration process could be application driven. See an example on Figure 7.
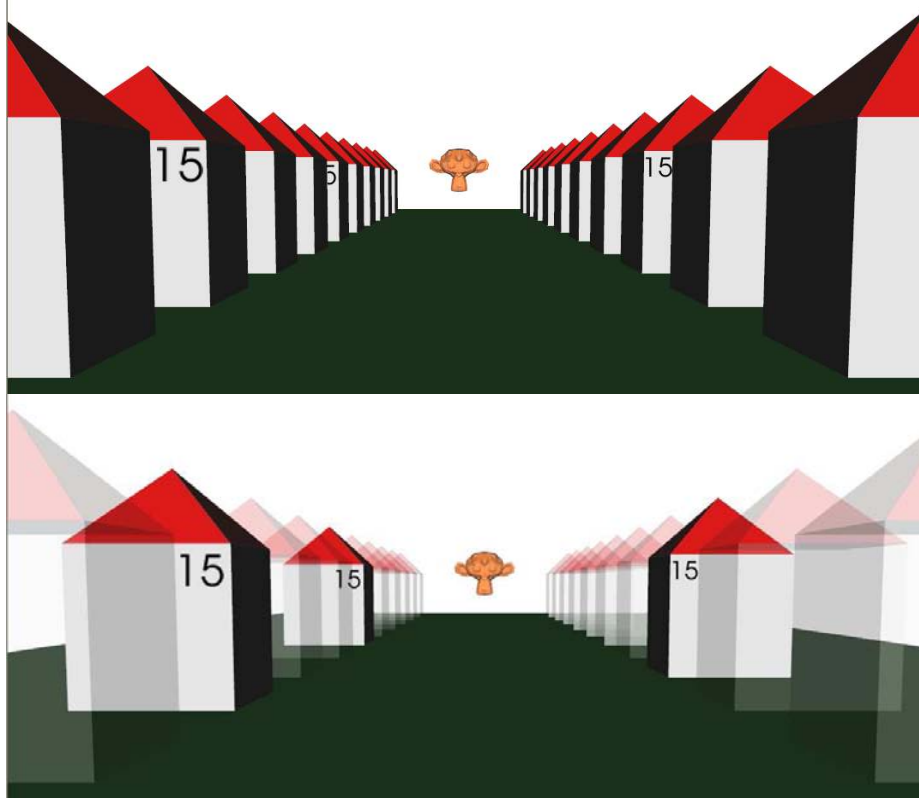
Figure 7. Some of the houses are described in semantic description as houses from 15th century. The visualization mode of these houses can be changed to highlight or suppress them.

Acknowledgements

References

[1]   Cmolik, L. Transformation of 3D scenes to 2D for Mobile Environment. *Master Thesis*, CTU in Prague, Prague, 2005.

[2] Diepstraten, J., Gorke, M. and Ertl, T. Remote Line Rendering for Mobile Devices, In *Proceedings of IEEE Computer Graphics International (CGI)'04*, pages 454-461, 2004.

[3] Hekmatzada, D., Meseth, J. and Klein, R. Non-Photorealistic Rendering of Complex 3D Models on Mobile Devices, In *8th Annual Conference of the International Association for Mathematical Geology*, volume 2, pages 93-98, Alfred-Wegener-Stiftung, September 2002.

[4] Sanna, A., Zunino, C., Lamberti, F. A distributed architecture for searching, retrieving and visualizing complex 3D models on Personal Digital Assistants. In *International Journal of Human-Computer Studies*, 60 (5) pages 701-716, 2004.

[5] Scalable Vector Graphics (SVG) 1.1 Specification. Retrieved on 20th April 2005 from the World Wide Web: *http://www.w3.org/TR/2003/REC-SVG11-20030114/*

[6] Sedlacek, J. Collaborative SVG Editor, *Master Thesis*, CTU in Prague, Prague, 2004.

[7] Vainio, T., Kotala, O. Developing 3D information systems for mobile users: some usability issues. In *Proceedings of the second Nordic conference on Human-computer interaction*, pages 231-234. ACM Press, 2002.

[8] Web Ontology Language (OWL) Specification. Retrieved on 20th April 2005 from the World Wide Web: *http://www.w3.org/2004/OWL/*

[9] Zunino, C., Lamberti, F., Sanna, A. A 3D Multiresolution Rendering Engine for PDA Devices, In *SCI 2003 Proceedings*, volume 5, pages 538-542, 2003.