Rapid Processing of Frames Refreshment with Viewpoint Movement Based on OpenGL for 3D Walkthrough System

1 Guoyou SHI, 2* Peng CHEN, 3 Shuang LIU
1 College of Navigation, Dalian Maritime University, 1 Linghai Road, 116026, China
2 Department of Computer Science & Technology, Dalian Neusoft Institute of Information, 8 Software Park Road, 116023, China
3 School of Computer Science & Engineering, Dalian Nationalities University, 18 Liaohe West Road, Dalian Development Zone, 116600, China

Received: 23 April 2014 /Accepted: 27 May 2014 /Published: 31 May 2014

Abstract: How to refresh frames for 3D scenes efficiently with viewpoint movement is very important for a 3D walkthrough system. Detailed calculation process of movement position and destination position for the viewpoint in virtual 3D scenes is described, thus implementing controlling scenes through the application of an external device. Moreover, OpenGL Performer pLOD node is utilized to process LOD (Level of Detail) display for improving scenes refreshment performance. With OpenGL's powerful interaction functions, 3D scenes are rendered with a fixed frame rate over 50 Hz. Then, details of MFC framework initialization and OpenGL initialization are given for Dalian nationalities university 3D walkthrough system. Running results of the software show the algorithm designed in this paper is feasible and effective and can meet requirements of real application. Copyright © 2014 IFSA Publishing, S. L.

Keywords: Viewpoint movement, Walkthrough system, OpenGL, LOD, Frames refreshment.

1. Introduction

With rapid development of computer graphics and virtual reality (VR), 3D walkthrough system is becoming more and more popular [1-7]. To provide powerful interaction functions in 3D walkthrough systems [8-12], efficiency of scenes refreshment with viewpoint movement is very important, especially running on a common personal computer with a standard graphics card and average amount of system memory. Because of the limited computational resources and common graphics card, it's a great challenge to realize the similar effect of smooth roaming operations in 3D environment on the platform of high performance computers and GPU hardware. Over the years, various methods of selection and interactive model creation are explored, as well as the design and implementation of an application able to create interactive objects from within an immersive virtual environment in real-time [13-15].

Providing sense of immersion and quick interaction responses are the basic of 3D virtual environment. There is a need to promptly affirm the
viewpoint, refreshing the corresponding scenes without any delay. In this paper, we present an algorithm that enables these desired operations by computing the 3D coordinate of a point on a surface-rendered image. Based on the researches of the characteristics of scenes refreshment, we used technologies of quick viewpoint movement computation and refresh operations to improve the efficiency of 3D rendering. And by taking full advantages of LOD processing functions provided by OpenGL, we rendered 3D scenes at real time rate. Finally, based on the methodologies, we implemented an experimental application system called Dalian nationalities university 3D walkthrough system to prove the high efficiencies of roaming operations.

The paper is organized as follows. Section 2 discusses fundamental theory analysis and detailed computation process for viewpoint movement. Section 3 introduces 3D rendering process and its implementation details based on OpenGL. Section 4 gives programming details of MFC framework initialization and OpenGL initialization for our 3D walkthrough system. Section 5 summarizes the paper.

2. Efficient Viewpoint Movement Computation

Viewpoint movement refers to frames refreshing regularly with input information of external devices. A prerequisite for frames refreshment is that scenario frames can be refreshed either in active mode or passive mode. Here, we can also explain viewpoint movement as different visual angle for the same building or object. Since viewpoint movement is based on input changes of external devices, we use mouse movement as an example to illustrate detailed process. Here, ‘W’, ‘A’, ‘S’, ‘D’ represents forward, backward, left, right movement. And we set assumption that mouse movement controls direction of visual angle directing.

2.1. Analysis of Viewpoint Movement

Generally, viewpoint will move forward or backward on direction of visual angle directing when ‘W’ or ‘S’ is pressed and it will move left or right on perpendicular plane of visual angle directing when ‘A’ or ‘D’ is pressed. Direction decision of visual angle depends on mouse movement. So when moves a viewpoint and decides direction of looking, it is crucial to decide vector \( \vec{d}t \) on direction of viewpoint movement and vector \( \vec{m}t \) on direction of mouse movement.

A vector can be decided by its start point and end point. Start point \( \vec{distart} \) of vector \( \vec{d}t \) is the position of viewpoint in 3D space, while end point is dynamic due to continuous changes of visual angle. At the same time, the start point of vector \( \vec{m}t \) is point \( \vec{dtend} \) and end point of \( \vec{m}t \) is decided by direction of mouse movement. In module of direction control for mouse, our programs ensure that point \( \vec{dtend} \) is located on a sphere centered at point \( \vec{distart} \) with fixed length radius. When viewpoint moves, vector \( \vec{d}t \) is not changed and point \( \vec{distart} \) is computed by external devices and vector \( \vec{dt} \). When mouse moves, 2D vector \( \vec{mx} \) is decided by forward position and backward position of mouse movement while mapping plane \( CW \) into visible client area. Vector \( \vec{m}t \) is computed by point \( \vec{distart} \), \( \vec{dtend} \) and vertical upward unit vector \( \vec{mu} \), so \( \vec{mtend} \) is obtained.

New vector \( \vec{dt} \) is started with \( \vec{distart} \) and ended with \( \vec{mtend} \), which is also the looking direction after mouse movement. What is noticed here is end point of vector \( \vec{dt} \) lying on circle profile. When \( | \vec{dt} | \) is larger than sphere radius, it is not desirable to do nothing because \( | \vec{dt} | \) will become infinite. So \( \vec{dtend} \) needs to be recomputed after computation of vector \( \vec{dt} \). Detailed information is shown in Fig. 1.

![Fig. 1. Computation diagram for viewpoint movement.](image)

2.2. Computation Process

We set the following variables as already known values in Table 1 when specific computation process is implemented. The following variables such as vector \( \vec{dt} \) on direction of viewpoint, circle profile \( CW \), tangent vector \( \vec{mt} \) starting from \( \vec{dtend} \) on circle profile \( CW \), end point \( \vec{mtend} \) and new viewpoint position \( \vec{dtend}^\prime \) need to be computed based on known variables.
Computation process follows four steps listed below.

Step 1, computes viewpoint direction according to start point and end point.

$$dt = dtend - dtstart = (a_1 - a, b_1 - b, c_1 - c)$$  \( (1) \)

Step 2, computes new start point \(dtstart\) and end point \(dtend\) when viewpoint moves on direction of \(dt\) with movement length \(S\) if ‘W’ or ‘S’ is pressed.

$$dtstart = (a_1 - a \pm a, b_1 - b \pm b, c_1 - c \pm c)$$
$$dtend = \left( \frac{2a_1 - 2a \pm a, 2b_1 - 2b \pm b, 2c_1 - 2c \pm c} {2} \right)$$  \( (2) \)

Step 3, computes vector \(x\) which is perpendicular to vector \(dt\) and \(m_{up}\) when ‘A’ or ‘D’ is pressed. In this case, viewpoint moves horizontally along the perpendicular straight line passing point \(dtstart\) on the plane which is determined by vector \(dt\) and \(m_{up}\).

$$\begin{align*}
(dt * x) = 0 & \quad \Rightarrow x = (c - c_1, 0, a_1 - a) \\
(m_{up} * x) = 0
\end{align*}$$  \( (3) \)

$$dtstart = (a \pm (c - c_1), b, c \pm (a_1 - a))$$
$$dtend = (a_1 \pm (c - c_1), b_1, c_1 \pm (a_1 - a))$$  \( (4) \)

Step 4, computes new destination position \(dtend\). Here, circle profile is tangent to circle with point \(dtend\) when mouse moves, so point \(dtend\) lies on plane \(CW\).

$$\begin{align*}
level &= \left( (b_1 - b) * 0 - (c_1 - c) * 1, (c_1 - c) * 0 - (a_1 - a) * 0, (a_1 - a) * 1 - (b_1 - b) * 0 \right) \\
&= (c - c_1, 0, a_1 - a)
\end{align*}$$

$$\begin{align*}
abreak &= ((b_1 - b) * (a_1 - a) - (c_1 - c) * 0, (c_1 - c) * (c - c) - (a_1 - a) * (a_1 - a), (a_1 - a) * 0 - (b_1 - b) * (c - c)) \\
&= ((b_1 - b) * (a_1 - a) - (c - c_1)) \end{align*}$$

$$\begin{align*}
dtend[0] &= dtend[0] + level[0] * mx[0] + abreak[0] * mx[1] \\
\end{align*}$$

Based on the above computation methods, we can get movement location and destination position for viewpoint easily. By making full use of powerful drawing function of OpenGL, scenario frames are refreshed at reasonable rate and tolerable response time. As shown in Fig. 2, new 3D scenes switch smoothly when a virtual walker roams along the road with viewpoint movement. In terms of scenes smoothness, we know the human eye requires at least 24 frames per second. In this case, a person will not fell choppy or jittery. After running our 3D walkthrough system for six months, we got average frame rate of over 50Hz, which means the display is updated 50 times per second ensuring the software running smoothly.

### Table 1. Known variables assumption for computation.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertical upward unit vector</td>
<td>( m_{up} )</td>
<td>( (0,1,0) )</td>
</tr>
<tr>
<td>Sphere radius</td>
<td>( R )</td>
<td>1 (normalized value)</td>
</tr>
<tr>
<td>Length for one viewpoint movement</td>
<td>( S )</td>
<td>1 (normalized value)</td>
</tr>
<tr>
<td>Start position of viewpoint</td>
<td>( dtstart )</td>
<td>( (a,b,c) )</td>
</tr>
<tr>
<td>End position of viewpoint</td>
<td>( dtend )</td>
<td>( (a_1,b_1,c_1) )</td>
</tr>
<tr>
<td>2D vector for vector ( m_{t} )</td>
<td>( mx )</td>
<td>( (fx,fy) )</td>
</tr>
</tbody>
</table>

**Fig. 2.** Viewpoint movement location and the corresponding changes of destination position.

### 3. 3D Rendering Process Based on OpenGL

OpenGL and OpenGL Performer provide functions of building visual simulation applications and virtual reality environments with rapid rendering rate and maximizing the graphics performance of any application. As our 3D walkthrough system requiring real-time visuals, free-running or fixed-frame-rate display, we utilized OpenGL powerful interaction functions to complete high-performance rendering.

As shown in Fig. 3, 3D scenes rendering implementation is based on OpenGL selection mechanism and interaction functions. This rendering process not only takes full advantages of OpenGL
interaction, but also assures efficiency of real-time rendering speed without refreshing any scenes on the screen.

### 3D Scenes Rendering Process

1. Clipping objects outside of rectangle window

2. Establishing corresponding relation between transformed coordinates and screen pixels based on viewpoint transformation

3. Setting the viewport matrix in function OnSize

4. Drawing process of 3D scenes

All 3D walkthrough systems have finite capabilities that affect the number of geometric primitives that can be displayed per frame at a specified frame rate. Because of these limitations, maximizing visual cues while minimizing the object count is very important. Level-of-detail (LOD) processing is one of the most beneficial tools available for managing 3D scenes complexity for the purpose of improving display performance.

The basic premise of LOD processing is that objects that are barely visible because they are located a great distance from the viewpoint. To save rendering time, objects that are visually less important in a frame can be rendered with less detail. The LOD approach to optimizing the display of complex objects is to construct a number of progressively simpler versions of an object and to select one of them for display as a function of range. The OpenGL Performer pLOD node contains a value known as the center of LOD processing.

For LOD transition blending, we adopted the following method. A transition per LOD switch is established rather than making a sudden substitution of models at the indicated switch range. These transitions specify distances over which to blend between the previous and next LOD. These zones are considered to be centered at the specified LOD switch distance. For example, if the distance is the near edge of blend zone, then LOD1 is 100% opaque and LOD2 is 0% opaque. If the distance is at the center of blend zone, then LOD1 is 50% opaque and LOD2 is 50% opaque. If the distance is the far edge of blend zone, then LOD1 is 0% opaque and LOD2 is 100% opaque. With pLODTransition() provided by OpenGL Performer, we limited the transition distances to be equal to the shortest distance between the switch range and the two neighboring switch ranges.

### 4. Implementation Details

In this section, we discussed our programming details including MFC framework initialization and OpenGL initialization. For 3D modeling data parsing, we have presented in another paper. The rough process is as follow. Based on the original 3ds data file provided by 3D modeling software of 3DS Max, we parsed these data and saved them with self-defined data structure. When 3D scenes are rendered, list and object classed is used to draw points and faces framework, while material library is used to render each scene to establish the virtual reality environment. In walkthrough process, OpenGL and OpenGL Performer drawing functions are utilized to implement real-time interaction with users.

Under MFC framework and OpenGL, we need to do some initialization work to finish 3D scenes rendering. After MFC initialization and OpenGL initialization, 3D scenes rendering process follows steps described in Section 2.

Before calling OpenGL functions provided by OpenGL library, we should do something initialization work for MFC framework. Detailed operations are presented in Fig. 4, including adding two window styles before creating window, setting pixel alignment format and setting device features and so on.

OpenGL initialization includes clearing the window, clearing depth buffer and color buffer, starting smooth coloring mode, starting depth inspection, starting illumination and light and so on. Details of calling function are presented in Fig. 5.
drug verification process.

The proposed approach in real applications.

The system verifies the efficiency and feasibility of the amount of main memory. Running effects of this computer with a standard graphics card and average called Dalian Nationalities University walkthrough also discussed. A complex walkthrough 3D system Programming details of a 3D walkthrough system are with users such as mouse movement or shortcut keys.

A selection approach is adopted for real time interaction powerful interaction functions, a reasonable 3D LOD frames refreshment method. Utilizing OpenGL movement, we presented a quick computation and

The 2014 Copyright ©, International Frequency Sensor Association (IFSA) Publishing, S. L. All rights reserved.

References

Fig. 4. Initialization process for MFC framework.

Fig. 5. Initialization process for OpenGL.

5. Conclusions

Based on detailed analysis of viewpoint movement, we presented a quick computation and frames refreshment method. Utilizing OpenGL powerful interaction functions, a reasonable 3D LOD selection approach is adopted for real time interaction with users such as mouse movement or shortcut keys.

Programming details of a 3D walkthrough system are also discussed. A complex walkthrough 3D system called Dalian Nationalities University walkthrough system is reconstructed on a common personal computer with a standard graphics card and average amount of main memory. Running effects of this system verifies the efficiency and feasibility of the proposed approach in real applications.

References


[10]. J. F. Ma, S. L. Bao, An algorithm for efficient
determination of the 3D coordinates for a spatial point selected on a 2D surface-rendered image, in Proceedings of the International Conference on Computer Science and Service System (CSSS), Nanjing, China, 2011, pp. 2025-228.


