OPEN INVENTOR EXTENSIONS

Jaroslav Přibyl
Doctoral Degree Programme, FIT BUT
E-mail: pribyl@fit.vutbr.cz

Jan Pečiva
Doctoral Degree Programme, FIT BUT
E-mail: peciva@fit.vutbr.cz

Supervised by: Pavel Zemčík
E-mail: zemcik@fit.vutbr.cz

ABSTRACT

This paper presents Open Inventor extensions for working with offline divided models over a network. There are in fact two extensions. The first one adds a functionality for downloading models and textures from a network. There is allowed a non blocking texture and model downloading. The second one extension adds a functionality for maintain level of detail for downloaded textures. The main purpose for maintain level of detail for textures is in saving maximum free memory. Open inventor is object oriented library for interactive working with 3D grapics and is based on synchronous scene graph traversal. On the other hand these two extensions is asynchronous. We also present a possibilities to synchronize these extensions and the scene graph traversal.

1. INTRODUCTION

The main motivation for extending the Open Inventor library was system for interactive examining large models stored on network server. We decided to use The Open Inventor library and then design its extensions. The design of these two extensions is strongly affected by the Open Inventor library design and principles. The system should have to be small and flexible. It means that the system should visualize data with achieving maximal interactivity.

The system download all needed data from network. Data downloading and textures downloading was designed as a non blocking process. This is the first extension. The second extension maintains level of detail for textures. The reason for implementing that feature is to minimalize used memory . Concurrently we want achieve maximal detail in visible part of the scene. Because we want to achieve more free memory, we have to store some texture on disk instead on memory. This feature unfortunately disagree with interactivity requirements.

2. DOWNLOAD MANAGER

The download manager will be used only for downloading models and textures. Our objective subsists on implementing the manager and synchronize it with the scene graph traversal.

2.1. BASIC CONCEPT

We have to decide how to hook up the whole download process into scene graph. There are at least two possibilities how to do it. First we can create new node with ability to download files. This node will process all requests from all his children nodes. Second we can create global download manager. So all nodes in scene graph send requests to this global manager. Compared with the first method, there we get control over all requests and we can better manage priorities of the requests with the second method.

We decided to implement the second method to take global care of downloading models and textures. It is much more easier to implement and it is without disadvantages compared to the first one method. The download manager simply receives download requests from all nodes and places them to a priority queue.

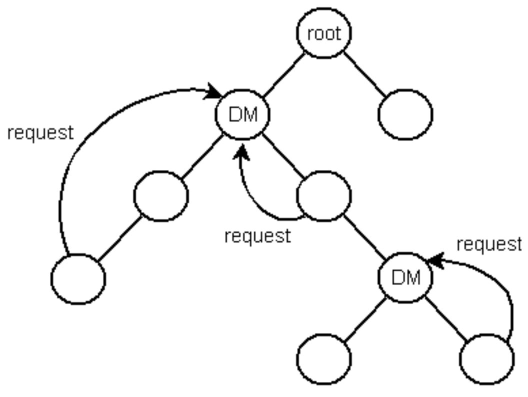

Figure 1: Each download manager DM cares about his children nodes.

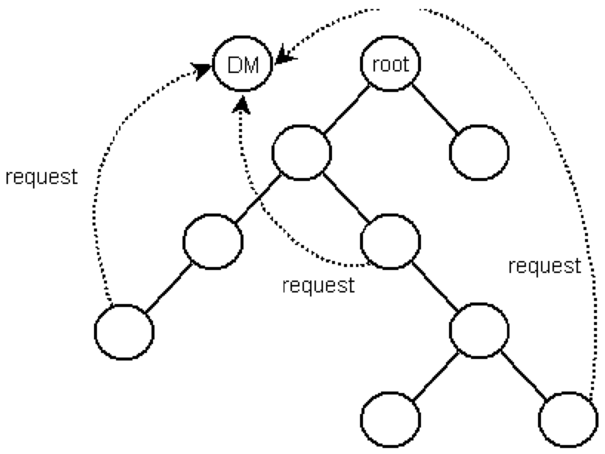

Figure 2: There are one download manager for all nodes in the scene graph.

2.2. JOIN WITH OPEN INVENTOR

As we said, there are the download thread and the priority queue, which takes care of the nodes‘s requests. This priority queue takes the highest priority request (queue head item) and download them using a download thread. After a model or texture is downloaded, the download manager calls the request‘s source node callback. This callback function notify the scene graph for redraw or rebuild.

The download process (thread) take only a minor processor time, so there are no performance issue concering to a frame rate. The node‘s download requests can be in a non blocking fashion, because the download thread.

3. LEVEL OF DETAIL FOR TEXTURES

There it is assumpted that we will use large models with a number of large textures. It means we will use textures on high resolution e.g. 2048 or 4096 pixels. On todays hardware there aren‘t possible to load all these texture data (many shapes => many hi-res textures) into memory. So we decided to use level of detail for textures. This feature allow us to save a lot of free memory space along with achieve maximal detail in user‘s view frustrum.

3.1. Basic concept

We have to decide how to select the appropriate level of detail for texture. The simplest way how to do it is in accordance to distance between the viewer and the given shape (texture). But as you know doing only this is in fact a bad way. The better solution is based on mapping coordinates from screen space into model space, see figure 3. and figure 4.

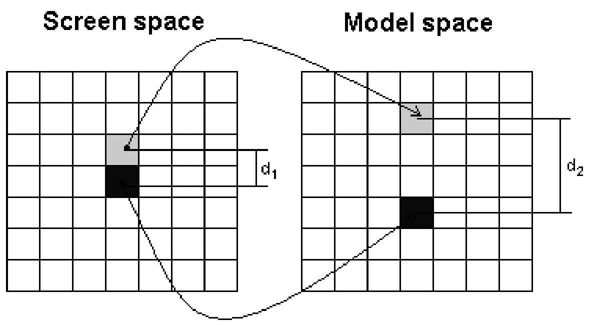

Figure 3: By projecting of two adjacent points from screen space to model space we can determine an apropriate resolution.

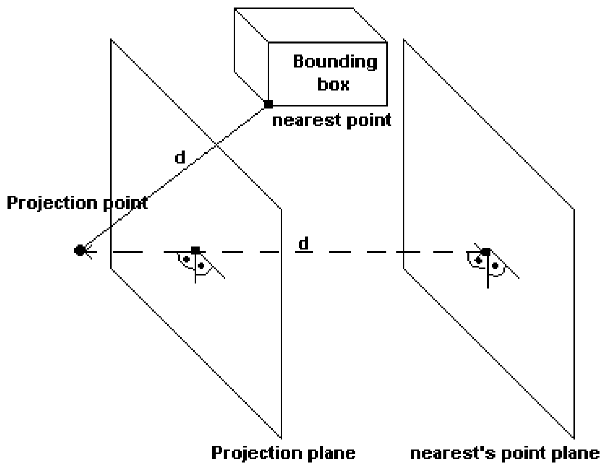

Figure 4: The two adjacent points from screen space are projected on plane in model space.

The nearest point‘s plane is parallel to projection plane and distance from projection point is defined by distance from projection point to the bounding box nearest‘s point. It means that texture level change only with varying distances and has same level in centrical circles. We compute the appropriate level of detail only by dividing the d_2 and the d_1 distance. We can multiply the viewport resolution by this division and get a new resolution which should be used. Well, we need to know input resolution of all textures, because right select the appropriate level according newly computed resolution. All what we need to know is input resolutions, no distance and no more additional informations.

3.2. Join with Open Inventor

At now we know how to select level of detail for textures. But this is only first step for success. We have to made the second step too. The second step rests in joining the texture level of detail technique with Open Inventor.

Texture switching can be divided into two separate problems. First we have to load texture from disk (generally from external source) and if it isn‘t a raw data, we should decompress it too. Second we have to bind the texture into opengl, create mipmaps etc. We also implement a separate loading thread for texture reading from disk and to decompressing it. Settings up the texture parametres, binding the texture or generating mipmaps can be done by the main redering loop (thread). So after the loading thread prepare texture data, it‘ll use a callback to invalidate the present texture data. This can be done simply by setting the texture invalidate flag in Open Inventor. For large textures we unfortunately met a performance issue (framerate fluctuating during any smoot animation e.g. model rotation). In next chapter we will discuss how to reduce this annoyingly framerate fluctuating.

4. DATA STREAMING

As we said, for large textures we met a performance issue. There are two problematic places in texture switching process. First is the loading/decompressing process and second is binding the texture into opengl which is done in the main rendering thread. We made some measurement and found out that binding large textures into opengl is much time consuming operation. So more info about data streaming you can found at [2].

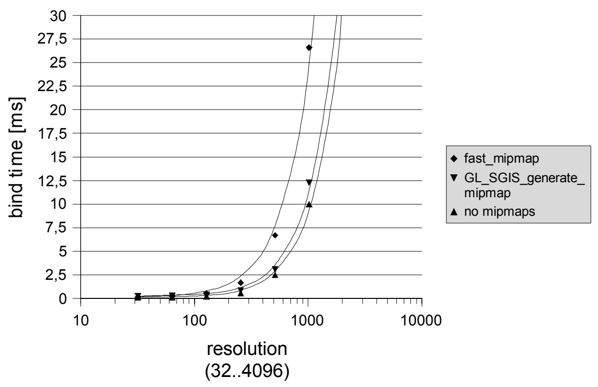

Figure 5: Texture binding into opengl measurement from 32 to 4096 resolution.

Resolution	Mip-map generation method [ms]		
	fast_mipmap (software)	GL_SGIS_generate_mipmap	no mipmaps
32	0,15	0,24	0,12
64	0,22	0,28	0,14
128	0,51	0,40	0,22
256	1,66	0,84	0,57
512	6,68	3,05	2,51
1024	26,58	12,29	9,99
2048	107,86	49,24	40,50
4096	571,60	379,65	262,9

Table 1: This table shows realation between texture resolution and texture creating/binding time.

Well, we had examined a several methods how to avoid the frame rate fluctuating. First we have to be able to load and decompres texture from disk to memory without affect the framerate. It can be done using a loading (streaming) thread. There are at least three ways how to implement such a thread, see figure 6.-8.

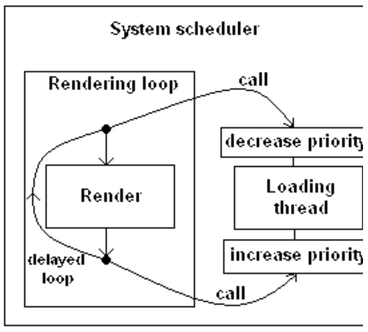

Figure 6: Actually we use this method. This method slightly affect the framerate.

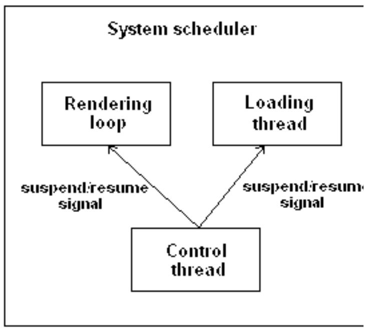

Figure 7: This method don‘t affect framerate, but we face up deadlocks (can be eliminated) and other issues. And this is not well designed solution too.

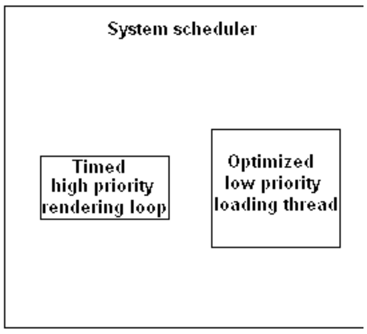
System scheduler
Timed high priority rendering loop
Optimized low priority loading thread

Figure 8: This is a best concept, but practically it is offen hard to optimize the workload for the loading thread.

So lets assume that the loading thread don‘t affect the interactivity (frame rate). If you look back on table 1. you will see that binding texture at 2048 or higher resolution into opengl is a time consuming operation. We uncovered, that working with resolution less or equal 1024 pixels didn‘t affect interactivity. But resolutions higher than 1024 is an issue.

5. BINDING LARGE TEXTURES

Binding large textures (greater than 1024) into opengl is time consuming operation. If you would save more free memory, you have to work with texture binding into opengl (no other way). It looks like a insolvable problem, but there are at least three approach how to attenuate the necessary binding time.

The first approach is based on binding texture into opengl per part using glTexSubImage2d function. This sounds like a good idea, but it is much more complicated to realize in Open Inventor. We would have to separate the image data and add a state flag into texture image class, which will specify the texture load status. As soon as the texture is loaded and whole binded into opengl, we can use it. For this approcah we made a simple demo and we should make a hard testing in near future.

The second approcah is based on second rendering thread. This second thread has its own rendering context. We are able to create a texture object in this second rendering thread and share this texture object with the main rendering context used in main rendering loop. As well as in first approach we made only testing application and next we should make a hard testing this method in near future.

The thirdth approach stands for reusing preexisting texture objects (opengl term). If we think that we have e.g. four texture objects on 2048 resolution permanently created and we only change the texture data, it would rapidly increase preformance, because we can only render to texture (direct to graphics hardware or this objects). With creating texture objects for small textures there is no performance issue.

6. CONCLUSION

As you can see, creating system for working with large textures is very difficult task. As far as you have to achieve interactivity, it becomes almost unsolvable. But there exists some avenues of approach how to partially solve this task. An alternative approach consist in taking a think how to making the high resolution textures away. Is there possible to divide the 4096 texture into four parts each at 2048 resolution or into eight parts each at 1024 resolution? It might be a better way.

REFERENCES

[1] Wernecke J., Open Inventor Architecture Group, The Inventor Toolmaker: Extending Open Inventor, Brno, Addison-Wesley, 1994, ISBN 02-01624-93-1

[2] Stuart Denman, Highly Detailed Continuous Worlds: Streaming Game Resources from Slow Media, GDC, 2003

[3] Anders Brodersen, Real-time Visualization of Large Textured Terrains, University of Aarhus

[4] Borgeat L., Godin, G., Blais, F., Massicotte, P., Lahanier, C., GoLD: Interactive Display of Huge Colored and Textured Models, Proceedings of ACM SIGGRAPH 2005