Zoom Rendering: Improving 3-D Rendering Performance With 2-D Operations

Tomasz Mazuryk, Dieter Schmalstieg, Michael Gervautz
Institute of Computer Graphics, Vienna University of Technology
Karlsplatz 13/186/2, A-1040 Wien, Austria
email: lastname@cg.tuwien.ac.at

Abstract. We propose a new method to accelerate rendering during the interactive visualization of complex scenes. The method is applicable if the cost of per-pixel processing is high compared to simple frame buffer transfer operations, as supported by low-end graphics systems with high-performance CPUs or 2-D graphics accelerators. In this case rendering an appropriately down-scaled image and then enlarging it allows a trade-off of rendering speed and image quality. Using this method, more uniform frame rates can be achieved, and the dynamic viewing error can be reduced.

1. Introduction

Virtual reality systems allow people to interact with computers much more intuitively than any other kind of user interface paradigm. Users can experience immersive presence in 3-D virtual worlds generated by the computer, observe and evaluate non-existing models, interact with autonomous agents and even meet other users in a simulated surrounding.

Unfortunately these new wider horizons of human-computer interaction come at a price of system performance that is still hardly tractable with even the most powerful hardware. In particular, rendering hardware must be fast enough to support interactive frame rates, so that a feeling of immersion in the simulated environment is achieved.

The quality of interactive presentation is related to the number of images presented per second. In general, displaying at a higher frame rate creates a more convincing presentation. Besides, variations in the frame rate are disturbing and unpleasant for the observer, so maintenance of a constant or near constant frame rate is desirable. Ideally, the application should be able to display a very high frame rate, and present a new image at fixed intervals [1]. This is in general hindered by variations in the number of visible objects. A large number of objects may be visible from one viewpoint, while only a small number is visible from another. Simply rendering all potentially visible polygons will result in significant variations in frame rate.

Furthermore, immersive systems often exhibit substantial lag between user action and perceived system response. Lag is composed of the time needed to process the user's input, and the time needed to produce the image, up to the moment when it is displayed on the screen [2]. Ideally, lag would be zero, and system response immediate. In reality however, reading of tracking devices, data base processing, and the image generation itself consumes time. The human perception system is very sensitive to lag, in particular if the correlation between head movement and according viewpoint update is disturbed. Simulator sickness may be the consequence [3].

To reduce variations in frame rate and lag, we propose to trade image resolution for interactivity, if the image at full resolution is too complex to be handled efficiently. This is frequently the case on low-cost computer systems without special 3-D acceleration. The method can be easily added to existing rendering software, and also integrated with a level-of-detail algorithm. It can also be used to reduce dynamic viewing error in head-tracked systems.

2. Related work

In previous work, several methods have been proposed to reduce the time needed to generate images of complex scenes. These approaches can be categorized by the step of the rendering process that they try to optimize. The rendering process is composed of four main stages (fig. 1):

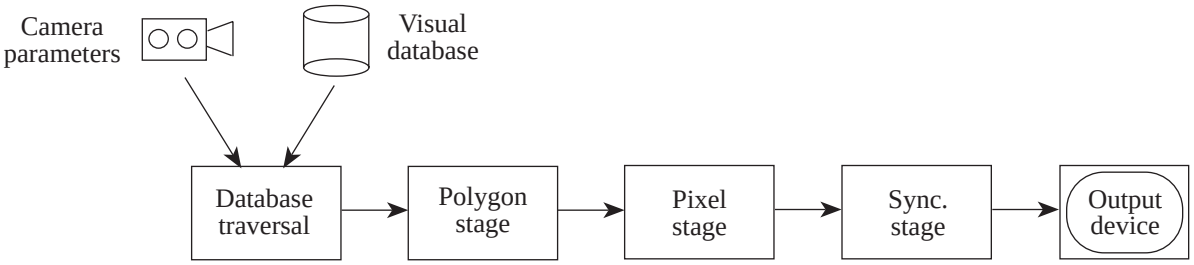


Figure 1: Stages of rendering process

- **Database traversal stage**: Database processing involves traversal of the scene, and passing the objects' polygons to the rendering hardware.
- **Polygon stage**: Polygon processing involves clipping, coordinate transformation, and lighting calculation.
- **Pixel stage**: Pixel processing involves scan conversion, Z-buffering, and texturing.
- **Synchronization stage**: After the image generation is finished, the process has to wait for the next vertical retrace to switch buffers and present the new image. During this time, the renderer sits idle. This delay can be quite substantial. An example is given in fig. 2: rendering time of frame number 2 is slightly longer than in case of frame 1 and 3, but the presentation time is delayed by additional 33%.

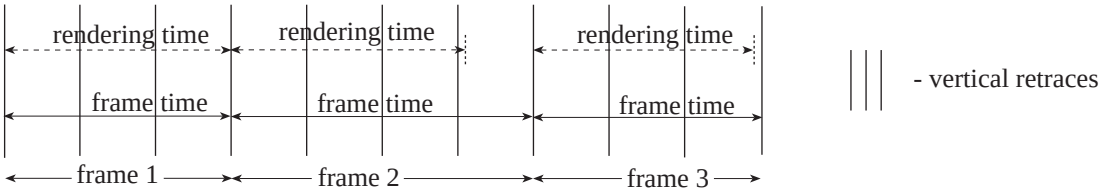


Figure 2: Time dependencies in the rendering pipeline

2.1. Database traversal stage

Visibility preprocessing can help to reduce the cost of database traversal. In a complex environment, big portions of the scene are occluded and need not be rendered at all. A precomputed data structure is used at rendering time for rapid determination of visibility. Teller and Séquin [4] describe a method based on potentially visible sets. They use a data structure composed of cells and portals that connect the cells. Greene et al. [5] claim to have a superior method that uses a combination of a scene-space octree and a Z-pyramid (a quad-tree like extension of a Z-buffer) to handle occlusion while performing front-to-back rendering. While both methods can bring a substantial data reduction, there is no upper limit on the amount of data passed to the polygon stage.

2.2. Polygon stage

Not every object in the scene is equally important for image appearance [6]. Image fidelity can be adaptively traded for rendering time. Timely presentation of a lower-fidelity image is better than late presentation of a higher-fidelity image. This observation provides the foundation for level-of-detail methods, that reduce the rendering time by choosing coarser resolutions of objects over finer resolutions, if the expected quality degradation is negligible. To provide an opportunity for a choice of fidelity, an object is represented by multiple resolution or *levels of detail*.

A very sophisticated usage of this idea is presented in [1]: A constrained optimization algorithm is used to choose a level of detail for each object to generate the "best" image possible within the target frame time. The major improvement over previous methods is that near-constant frame rates can be guaranteed.

2.3. Pixel stage

Funkhouser and Sequin's method also includes the pixel stage: For every object, not only a level of detail but also a rendering method is selected according to the desired fidelity. In our opinion, this is only effective if there is a real difference in the time needed to perform different rendering methods (e.g. flat shading or Gouraud shading). This is certainly the case if the shading is computed by the CPU. If shading hardware is available, however, using a cruder shading simply disables one or more functional units, and throughput is usually only marginally increased.

2.4. Synchronization stage

A very radical idea reduces the time for the synchronization stage to zero by discarding the concept of a frame. Usually double buffering is used to avoid the artifacts associated with updating an image while it is presented. Bishop et al. [7] present an alternative they call *frameless rendering*. The method uses a single buffer, and updates single pixels in random order. This method avoids the need to wait for the vertical synchronization, but is makes use of image and object coherence impossible. Thus it can only be used with renderers based on raytracing, and is not compatible with today's image generators.

3. Overview of the approach

3.1. Zoom rendering

Our approach aims at an optimization of the pixel stage and the synchronization stage. If the pixel-processing itself cannot be improved, only a reduction in the number of pixels can speed up rendering. A smaller image contains less pixels, so we generate a down-scaled version of the final image, effectively undersampling the target image in the 2-D domain. This image is then enlarged using pixel replication by integer factors (*zoom rendering*). The upscaling is a simple memory copy operation that takes negligible time compared to the expensive geometry processing. This speed-up comes at a price: Because the approach is based on undersampling, aliasing is introduced in the form of "blocky" resolution. The disturbing effect can be greatly reduced by applying a fast blurring filter such as described in [8] and [9], and the result can hardly be distinguished from an image rendered at screen resolution, especially if the viewpoint dynamically changes.

If the pixel stage is the bottleneck that causes low frame rates, this approach improves overall throughput by greatly reducing the number of pixel to be processed, and allows higher frame rates. If a hardware-assisted graphics pipeline is being used, this is particularly important, because the slowest stage determines the overall system throughput.

3.2. Frame synchronization

Zoom rendering needs a minimum of 2 buffers: One for rendering the smaller image, and one to hold the zoomed image presented to the user. It should be noted that this is not the same as double buffering, which also uses two buffers (one for rendering, one for displaying), but switches buffers during the vertical retrace phase. In zoom rendering, buffer switching is replaced by the copy/zoom operation. This assumes that the memory copy operation used for zooming is fast enough, so that no disturbing effects can be perceived, although the target image is visible during the update. This is the case if hardware-assistance for raster graphics is available, such as fast memory copy provided by some CPUs and graphics controllers.

Therefore zoom rendering does not waste time waiting for synchronization. Updates are presented to the user immediately. Although this exposes variations in the frame rate directly to the observer, it is a big advantage, because small variations in rendering time cannot lead to the loss of a whole screen refresh cycle (compare fig. 2). The small variations can barely be noticed, while the loss of a screen refresh cycle can significantly hurt frame rate.

3.3. Evaluation of graphics hardware

A broad range of image generators is used today, ranging from simple frame-buffer boards to dedicated pipelined 3-D processors. The optimal choice of rendering optimization method is dependent on the capabilities of the target system. We examine typical categories of graphics systems, and discuss how well zoom rendering performs on them.

- **Low-end**: No special graphics hardware is available except for a simple frame buffer. All 2-D and 3-D graphics operations have to be performed by the CPU. Examples for such configurations are the 8-bit entry graphics option for SGI Indy workstations or standard VGA adapters for PCs. Zoom rendering is useful if memory copy operations can beat per-pixel processing. Because pixel processing (e.g. Z-buffer calculations) are more complex operations than memory copying, zoom rendering improves performance for this machine class.
- **Mid-end**: Mid-range graphics boards are often equipped with special chips capable of bit block transfer (“BitBlt”) operations such as fast filling and copying of rectangular regions of the frame buffer. BitBlt operations are essentially 2-D functions designed for the use in windowing systems operating with two-dimensional images. These 2-D accelerated boards are often found in desktop computers (“Windows-accelerator” cards for PCs). The BitBlt operations make the zoom rendering extremely fast compared to the per-pixel processing. Pixel-processing is a 3-D operation and therefore has to be carried out by the CPU, while zoom rendering uses the special 2-D hardware support.
- **High-end**: High-end graphics systems provide dedicated processors for 3-D operations. These can include clipping, coordinate transformation, lighting, rasterization, visibility determination (Z-buffering) and texturing. The most prominent example is the Reality Engine. With such hardware support, pixel processing is very fast so that zoom rendering usually does not improve rendering time.

In summary, zoom rendering is most beneficial if 2-D operations are fast compared to 3-D operations. This is true for what has been categorized as “low-end” and “mid-end” above. It should be noted that the market for “standard” graphics is currently moving towards 2-D accelerated graphics, driven by the strong demand for graphical interaction and windows-based systems.

4. Application of the method

4.1. Improvement of uniform frame rates

Zoom rendering is a method that trades off image quality against speed of image generation (image degradation). In that way it is very similar to level-of-detail methods, and the same

considerations apply. Adaptive image degradation methods choose the level of degradation at runtime, and attempt to achieve a constant or near-constant frame rate. The input needed to determine what image fidelity to choose can be based on one of the two following principles:

- **Reactive fidelity selection**: The chosen fidelity is based on the complexity of the previous frame. This assumes that there exists coherence in complexity between subsequent frames, so that the previous frame is a good estimate for the current frame. However, this approach can fail if the number of visible object suddenly changes.
- **Predictive fidelity selection**: If a good prediction of the current frame's complexity is available, it can be used to select the fidelity. In contrast to the reactive approach, a sufficiently accurate prediction can guarantee a bounded frame time.

We see three possible configurations how zoom rendering can be embedded into an image generator:

- **Reactive zoom rendering**: Zoom rendering is added to an existing renderer with minimal changes. The zoom factor is chosen based on the last frame's time.
- **Predictive zoom rendering**: If sufficiently detailed information about the hardware capabilities are available, the zoom factor is chosen by predicting the time available for pixel processing of the currently visible geometry. This of course also requires details about visibility.
- **Integration with a level-of-detail algorithm**: Zoom rendering may be viewed as another image quality vs. time trade-off. It can therefore be integrated with an existing level-of-detail algorithm as another measure to save rendering time. Zoom rendering operates on whole images, whereas level-of-detail selection operates on scene objects, but this should only slightly increase the algorithm's complexity.

4.2. Dynamic viewing error reduction

Another application for zoom rendering is in the field of viewing error reduction, as needed for head-tracked systems. Head-tracking is used to adjust the viewpoint according to the user's head movements. Lag creates viewing errors, because the image presented to the user does not correspond to the user's actual head position. As already mentioned, the lag is mostly due to the time needed for rendering.

Prediction can be used for viewing error reduction. A predictor (often based on Kalman-filtering) is fed with the most recently measured data, and produces a prediction of future measurements. The predicted value is then used instead of the measured value to define the viewpoint used for image generation. If the prediction interval matches the time needed to produce the image and prediction is correct, the predicted head position corresponds to the actual head position at the time of image presentation. However, none of the existing prediction algorithms is perfect, so a disturbing amount of viewing error is left. This error is only tolerable for short prediction intervals, so the use of prediction is still bound to efficient rendering that creates only little lag [2].

In order to keep the prediction interval small, it is still desirable to keep rendering time as short as possible. Especially if the user is moving the head very rapidly, accuracy of prediction is compromised, so that the viewing error may become unacceptable. The situation is further complicated by the fact that rapid head movements are often used to orient oneself, so correct position of the objects on the screen is crucial.

Zoom rendering can be used for adaptive acceleration of image generation in such situations. Rapid head movements will result in the selection of a larger zoom factor, so that images can be generated and presented at a higher than average frame rate. The prediction interval is defined by the frame rate, so a faster frame rate means a shorter prediction interval, which in turn gives more accurate prediction results. This means that the predicted position is closer to the actual position, so the accuracy of the presentation is increased. In that way zoom rendering helps to dynamically reduce the viewing error. The downside is that zoom rendering sacrifices image resolution for speed. However, during rapid head movements, perception is very limited, and a low image resolution will not be very disturbing.

An additional benefit comes from the combination of zoom rendering and image deflection [2]. The idea of this approach is quite simple: A new frame is rendered with the predicted camera position/orientation values. When the new picture is ready in the backbuffer, the tracker is read again and the image is fitted according to the new tracker data. This can be done by simple 2-D operations (panning and scrolling). To perform the image deflection, the image has to be copied between buffers, so the zoom operation (which is essentially a copy operation) comes for free. A small amount of time can even be won because zoom copying frequently involves a small buffer, while standard image deflection always uses a full-size buffer.

5. Implementation

The rendering of one frame using the proposed zoom algorithm is performed in five steps as follows:

1. Choose zoom factor (based on graphics load, LOD algorithm, head movement speed etc.)
2. Set output viewport size according to the zoom factor and output window size (see also fig. 3a)
3. Render the picture to the viewport (which is invisible to the user)
4. Synchronize (choose the frame presentation time) - note that this does not have to be at the vertical retrace!
5. Copy image to the output window using the zoom factor calculated in step 1 (see also fig. 3b)

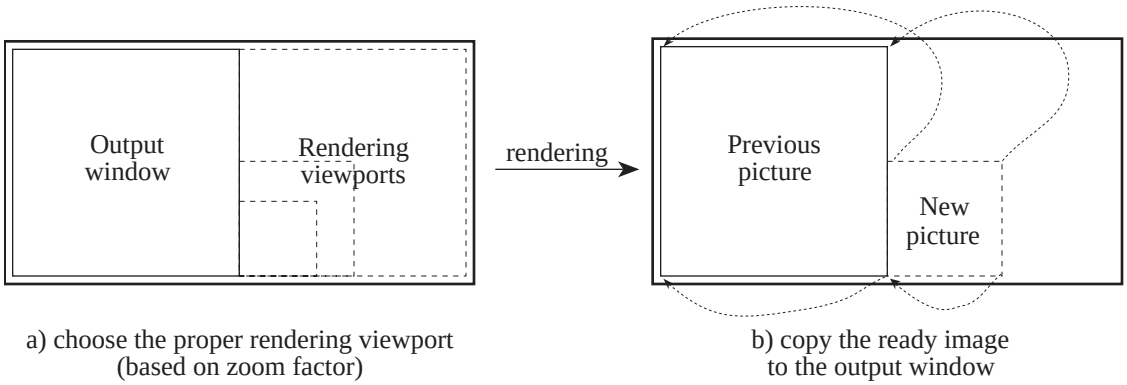


Figure 3: Zoom rendering steps

6. Results

The rendering method as described above was implemented in C++ and GL. The application was tested on Silicon Graphics workstations with a variety in performance:

- **Indy8**: SGI Indy (R4000/100 MHz, 8-bit non-accelerated graphics)
- **Indy24**: SGI Indy (R4000/100 MHz, 24-bit non-accelerated graphics)
- **Iris**: SGI Personal Iris 4-D/35 (R3000/36MHz, GP1.2 graphics)
- **Onyx**: SGI Onyx (2 x R4400/150MHz, RealityEngine2 graphics)

These machines cover a broad range of system performance. The Indies have fast CPUs, but no graphics acceleration. They therefore gave the best results for the application of zoom rendering, because the CPU and thus memory-bound CPU operations are relatively fast compared to non-accelerated 3-D operations. The Iris has moderate 3-D acceleration (shading and Z-buffering hardware), but a slower CPU. Consequently we expected zoom rendering to be less effective, which was confirmed by our measurements. Finally, we ran our test on an

Onyx with 2 fast CPUs and high-end graphics. 3-D performance on this machine is so fast that frame rate does not seem to be dependent on the number of pixels at all.

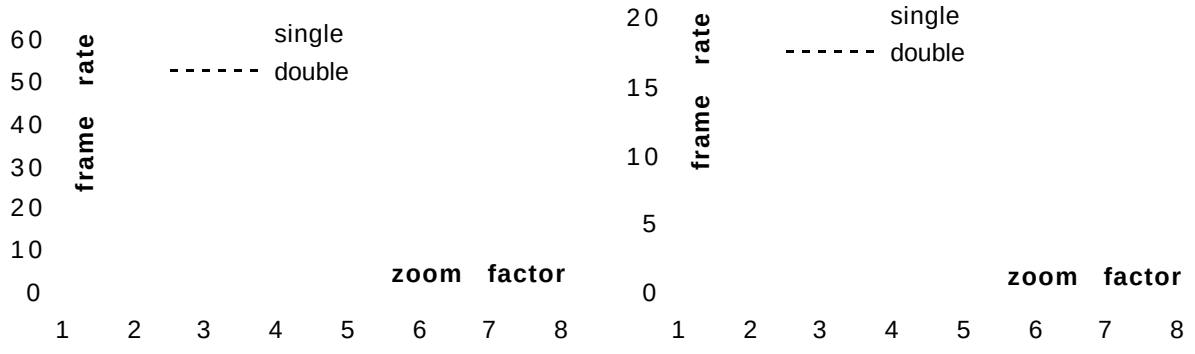
We evaluated zoom rendering on these machines with seven test scenes: one “realistic” scene (the model of an office, see also color section), and five representations of a sphere, tessellated with an increasing number of primitives. The fifth, most complex sphere representation was tested on the Onyx only, and with two shading methods: Gouraud-shading with precomputed colors (G1), and Gouraud-shading with colors that were computed in real-time from materials and lighting definitions (G2). G2 was constructed to saturate the polygon stage of the Onyx. Table 1 gives a quantitative description of the model.

Scene	#polygons	#vertices	shading
Office	145	580	flat
Sphere S0	20	60	Gouraud
Sphere S1	80	240	Gouraud
Sphere S2	320	960	Gouraud
Sphere S3	1280	3840	Gouraud
Sphere G1	5120	15360	Gouraud
Sphere G2	5120	15360	Real-time Gouraud

Table 1: Data on the test scenes used for evaluation of the method

The sphere test scenes only vary in the number of polygons that are used to approximate the sphere. The number of pixels is almost constant, because each of the Spheres S0-S3 are rendered at the same size, and all front-facing polygons are fully visible. Therefore the time for the pixel stage is solely dependent on the zoom factor, but independent of the sphere resolution.

We measured frame rate for zoom factors from 1 (no zoom) to 8 (number of pixels reduced by a factor of 64). At zoom factor 1, measurement was done without a memory copy step to allow a fair comparison of conventional and zoom rendering.



(a) Office scene (b) Sphere S3

Figure 4: Comparison of zoom rendering with single and double buffering on an 8-bit Indy

Fig. 4 compares the effect of using single buffering (non-synchronous frames) over double buffering (synchronous frames). The tests were taken on the 8-bit Indy for the office scene (fig. 4a) and the 1280-polygons Sphere (fig. 4b). The plots depict frame rate over zoom factor. For the office scene, we obtained an almost 6-fold speed-up with a zoom factor of 8.

The improvements can even be greater for simpler scenes, but scenes smaller than the office test case are not so commonly used.

The improvement of non-synchronized single buffering over double buffering is proportional to the frame rate. With double buffering, a frame's time has to be an integral multiple of the synchronization frequency (e.g. n·1/60 sec for a 60Hz display). Every small gain from zoom rendering brings noticeable advantage for single buffering, and scales linearly with the number of frames. For double buffering, however, the time won with zoom rendering has to exceed one synchronization cycle to be of use. This is unlikely for higher frame rates. Fig. 4a shows a 80% increase of single over double buffering for the simple office scene, while the improvement for the more complex scene in fig. 4b is only 20%. Note the asymptotic behavior of double buffering frame rates in both cases.

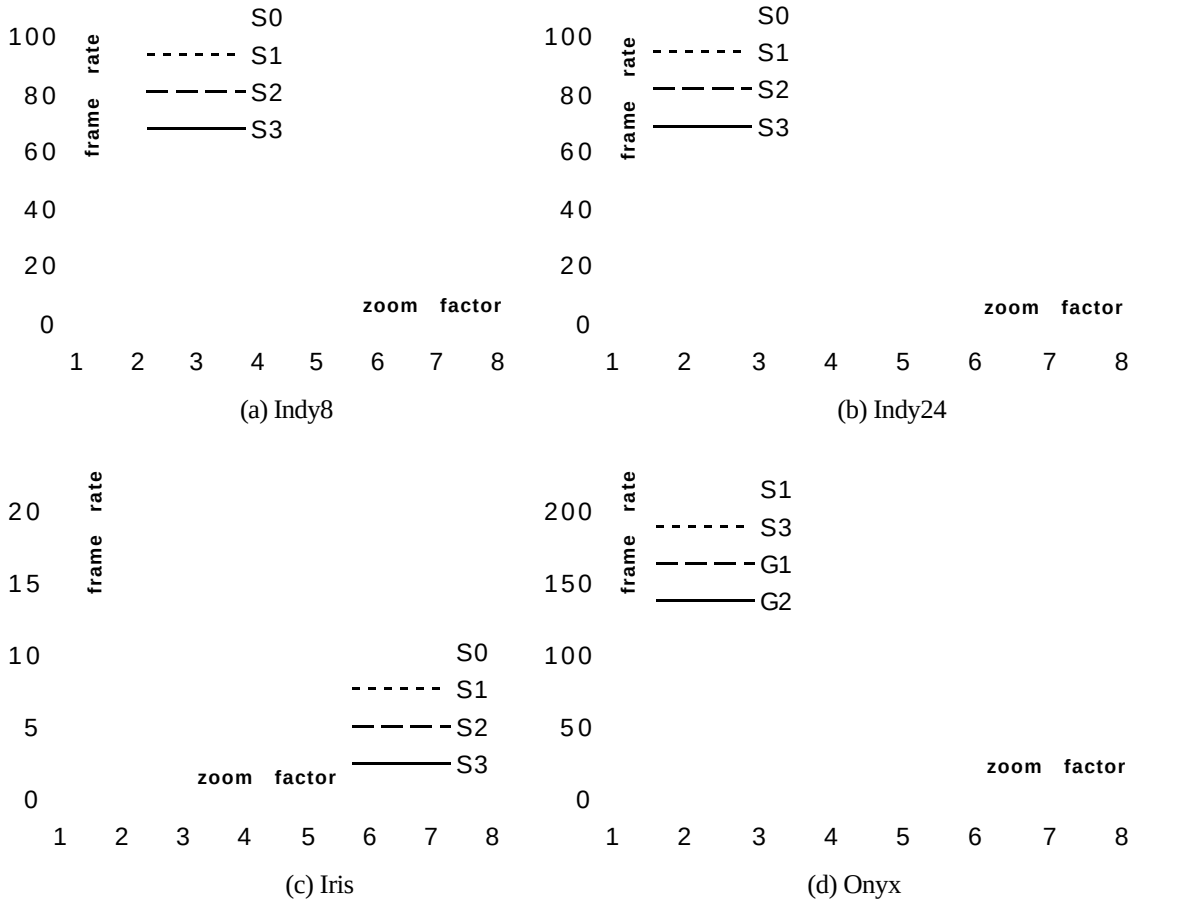


Figure 5: Comparison of zoom rendering performance for four different machines

Fig. 5a and 5b illustrate that the possible gains in frame rate are larger for simpler scenes. A complex scene with a large number of polygons spends most of the rendering time in the polygon stage, and optimizations of the pixel stage will bring much gain.

Note that the results for the 8-bit frame buffer variant are slightly better than for 24-bit, because less memory has to be handled. The crack at a zoom factor of 7 is apparently related to the way the CPU handles memory copying, which is particularly inefficient for this prime number.

Fig. 5c and 5d show measurements done on the Iris and Onyx, respectively. Both machines have 3-D graphics accelerator pipelines. For such machines, the slowest stage of the pipeline determines overall performance. If the polygon stage is saturated by a large number of polygons (i.e. complex scene or lots of lighting calculations), it becomes the slowest stage. In that case improvements in the pixel stage such as zoom rendering do not help. To demonstrate

this, we used additional, more demanding scenes on the Onyx (G1, G2). All curves have a slope of zero for higher zoom factors on the Iris (fig. 5c), and the same happens for most complex model G2 on the Onyx (fig. 5d). The crack in the curve at a zoom factor of 2 is caused by the introduction of the zoom/copy operation (no zoom/copy operation is used for full size rendering at zoom factor 1).

7. Conclusions and future work

We have presented a simple method that can help to achieve better results when visualizing complex environments at interactive frame rates. While visibility preprocessing and level-of-detail algorithms improve the database traversal and polygon processing stage of the rendering process, our method - zoom rendering - optimizes the pixel stage. A small image is rendered, thereby reducing the pixel processing cost, and then zoomed to the desired resolution. Blurring helps to minimize visual artifacts.

The method is most appropriate if 2-D acceleration (provided by fast CPUs or BitBlt chips), but no 3-D acceleration is available. Current desktop systems often fall into that category. The method can be added as an enhancement to existing renderers with little effort. It has also been successfully used for dynamic viewing error reduction of head-tracked systems. Future work will involve the integration of zoom rendering with a level-of-detail algorithm.

Acknowledgements

This work was sponsored by the Austrian Science Foundation (FWF) under contract number P11392-MAT.

References

[1] T. A. Funkhouser, C. H. Séquin: Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments. Proceedings of SIGGRAPH'93, pp. 247-254 (1993)

[2] T. Mazuryk, M. Gervautz: Two-Step Prediction and Image Deflection for Exact Head-Tracking in Virtual Environments. To appear in Proceedings of EUROGRAPHICS'95 (1995)

[3] J. L. Helman: Designing VR Systems to Meet Physio- and Psychological Requirements. SIGGRAPH Course: Applied Virtual Reality, No. 23 (1993)

[4] S. J. Teller, C. H. Séquin: Visibility Preprocessing For Interactive Walktroughs. Proceedings of SIGGRAPH'91, Vol. 25, No. 4, pp. 61-69 (1991)

[5] N. Greene, M. Kass, G. Miller: Hierarchical Z-Buffer Visibility. Proceedings of SIGGRAPH'93, pp. 231-237 (1993)

[6] P. Heckbert, M. Garland: Multiresolution Modeling for Fast Rendering. Proceedings of Graphics Interface'94, pp. 43-50 (1994)

[7] G. Bishop, H. Fuchs et al.: Frameless Rendering: Double Buffering Considering Harmful. Proceedings of SIGGRAPH'94, pp. 175-176 (1994)

[8] P. Rokita: Fast Generation of Depth of Field Effects in Computer Graphics. Computers & Graphics, Vol. 17, No. 5, pp. 593-595 (1993)

[9] K. Dudkiewicz: Real-Time Depth of Field Algorithm. Proc. European Workshop on Combined Real and Synthetic Image Processing, Hamburg (1994)

Keywords: zoom rendering, 2-D accelerators, BitBlt, level of detail, rendering pipeline