

A Highly Scalable Image-Based Remote Rendering Framework

Dipl.-Inf. Johannes Ghiletiuc

Dipl.-Inf. Markus Färber

Prof. Dr. Beat Brüderlin

Technical University of Ilmenau

Department of Computer Science and Automation

Computer Graphics Group

P.O. Box 100 565, D-98684 Ilmenau, Germany

Telephone: +49 3677 69 2767

Telefax: +49 3677 69 1285

johannes.ghiletiuc@gmx.de

markus.farber@tu-ilmenau.de

beat.bruederlin@tu-ilmenau.de

Abstract

We propose a method called Incremental Impostor Streaming (IIS) aimed at visualising very large 3D scenes remotely on low-end graphics hardware. An IIS server constructs an adequate image-based scene representation without the need for any pre-processing. Rendering quality scales neatly with bandwidth and client processing power.

The method features an adaptive impostor generation algorithm which reuses existing impostors and guarantees that an increase in bandwidth or processing power will always lead to better image quality. Impostors are created only at regions where rendering all existing impostors cause significant image depth differences with respect to the image generated by rendering the original scene. Edge detection on the depth values prior to mesh generation allows for effective handling of non-continuous 3D scenes. The maximum visibility error is bounded at any time. IIS is partly implemented directly on graphics hardware resulting in very fast impostor generation. Additionally, IIS can be implemented on top of almost any conventional graphics pipeline. The resulting scene representation is very compact and can be transferred quickly across most network links. IIS offers real-time rendering speeds and great responsiveness even on low-end clients with very limited graphics hardware.

Keywords

Real-time Rendering, Impostor Rendering, Remote Walkthrough, Streaming

1 Motivation and State of the Art

The ever-increasing amount of data to be visualised in CAD, research, education and entertainment requires new techniques that accelerate the rendering process. At the moment, data sets tend to grow faster than available graphics processing power. Mesh-based 3D models grow to sizes ranging from several gigabytes up to terabytes. Rendering these meshes using trivial methods in real-time is impossible, even on high-end hardware.

The same applies for transferring meshes across a network link. Downloading complete models would take far too long to allow for interactive viewing and editing. Even if the scene could be downloaded in reasonable time, rendering speed on most end systems is still limited. Therefore, minimising model size and thus transfer time is crucial for speeding up rendering. Of course, a minimised model must represent the scene as correctly as possible. There are many approaches which generate such representations. They all assume,

- that only a subset of the scene is visible at a given moment and
- that an approximate representation is sufficient for display.

Thanks to virtual camera characteristics and screen resolution limits both assumptions generally hold true. They give rise for many types of so called Level-of-Detail (LoD) algorithms.

A specific LoD method, impostor rendering, is able to derive an adequate scene representation that can be transferred much faster across local or wide area networks and can additionally be rendered very efficiently (independently from raw scene size). To capture scene depth complexity correctly, a flat impostor can be extended to a textured depth mesh (TDM).

Traditional LoD methods either lead to severe errors or require expensive pre-processing while common 2D screenshot transfer techniques suffer from lack of interactivity and responsiveness.

[DSS+99] propose a multi-layer approach that splits the scene into a local and a distant part. The distant part is then split up into layers which constitute a set of independent TDMs. For the most part, this approach relies on an offline pre-computing phase, which renders the technique impracticable for our needs.

In [Jes05], another layered impostors technique is presented. It provides concrete quality warranties with respect to common impostor artefacts like cracks, skins or blurriness. As long as the camera stays within a given viewing cell, artefact-free representation is possible, yet a lot of view cells must be pre-computed to handle a complex scene.

A more advanced approach utilising TDMs for impostor generation is presented in [WM03]. Using view cells as well, the ITDM (Incremental Textured Depth Mesh) algo-

rithm takes depth samples from multiple view points to create a skin- and crack-free triangle mesh representation. Unfortunately, taking and combining samples is costly and must be done in an offline pre-processing phase. Nonetheless, IDTM incrementally generates a TDM per view cell that can be displayed efficiently.

To provide a real-time capable method which is compatible with a typical 3D graphics pipeline, this work proposes a TDM representation integrated into a client-server framework, called Incremental Impostor Streaming (IIS). In contrast to existing techniques, it is streamlined to be used on-line, which means that expensive pre-processing is never needed – a key requirement for a remote rendering solution. IIS uses GPU-hosted algorithms. Specific design decisions provide options to directly steer impostor quality and/or size, thus situation-dependent TDM construction and visualisation guarantee interactive speeds.

The basic TDM idea is thus extended to a method that uses conventional rendering output to generate an adequate scene representation for almost arbitrary end systems and network links. This essentially means that additional bandwidth leads to better quality while leaving the rendering process independent from raw scene size.

The next section describes Incremental Impostor Streaming. Section 3 provides a more detailed explanation of the impostor generation process while Section 4 presents the achieved results, suggests future research and concludes.

2 Incremental Impostor Streaming Architecture

The client-server framework incorporates a powerful visualisation server, as proposed in [HPB05], which is the only place that has complete access to the raw scene. In most applications, very large models are generated on central servers anyway and clients have to connect to these servers to access the data for visualisation and editing purposes.

The Incremental Impostor Streaming server generates and manages an impostor database. Each impostor corresponds to a specific camera position and viewing direction; it can be thought of as a three dimensional screenshot. Because bandwidth and client rendering performance are limited, impostor generation is adaptive and optimises for rendering speed and display quality.

Figure 1 depicts the client-server architecture of the Incremental Impostor Streaming framework.

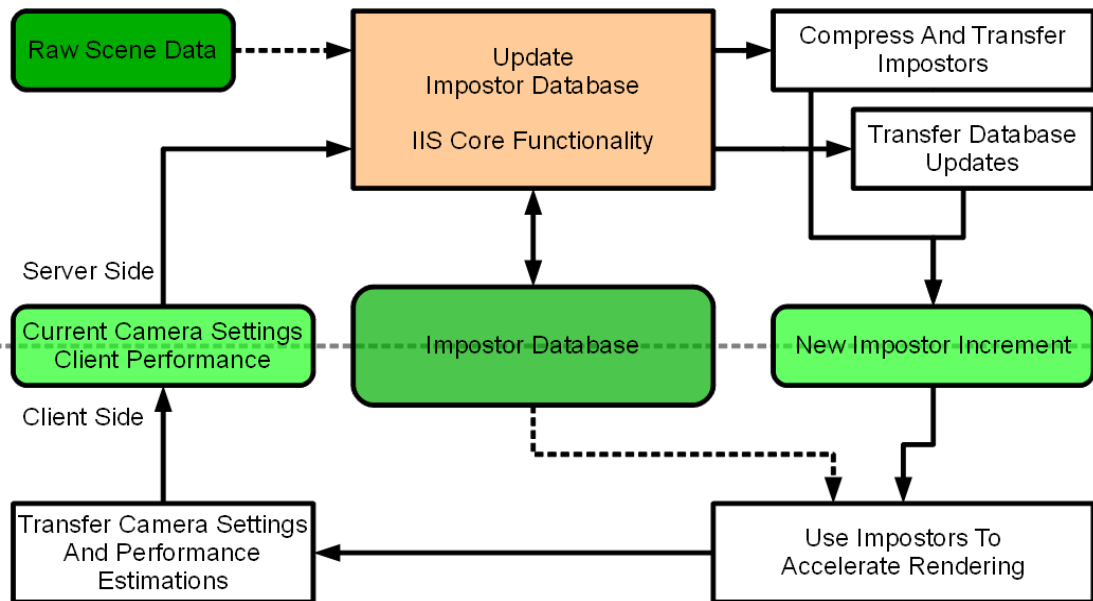


Figure 1: Incremental Impostor Streaming client-server architecture

2.1 Server Side

Only the server can generate new impostors. The server maintains a global impostor database and manages synchronisation of the database with database copies on the clients. Each impostor has the following components:

- Reference camera configuration (camera position, view direction and view frustum settings) as requested by the client
- A triangle mesh that resembles the scene depth structure as observed from the impostor's reference camera
- A texture (as mesh overlay) with colour data for each visible pixel

To determine adequate quality and camera settings to generate a new impostor, the server relies on performance measures and camera settings sent from the client. If the current database state exceed client-side limits (a triangle budget, for example), previously created impostors are removed and/or replaced by less demanding impostors. Eventually, the client is informed about the changed database state and newly created impostors are transferred across the network link.

2.2 Client Side

At the client the user visualises the 3D scene and therefore controls the current view. The client regularly sends the following parameters to the server:

- Camera position vector
- Camera direction vector

- Camera view frustum configuration
- Triangle budget for impostor database (usually a few hundred thousand polygons per frame for mainstream consumer 3D graphics hardware)

The client is solely responsible for rendering the current impostor database state. Depending on current camera settings, bandwidth and quality settings, the impostor database gets updated asynchronously by the server. At any given moment, the client renders all available impostors which are marked as valid. A conventional fixed-function rendering pipeline suffices to visualise the impostors. Impostor validity criteria are to be explained in detail in the next section.

3 Impostor Management – The Complete Cycle

As described above, Incremental Impostor Streaming core functionality is implemented at the server side. Figure 2 can be understood as a refinement of the “Update Impostor Database” block shown in Figure 1. In the following, all involved steps are described at greater detail. The cycle starts at the moment when the new camera settings and performance estimations sent by the client arrive at the server.

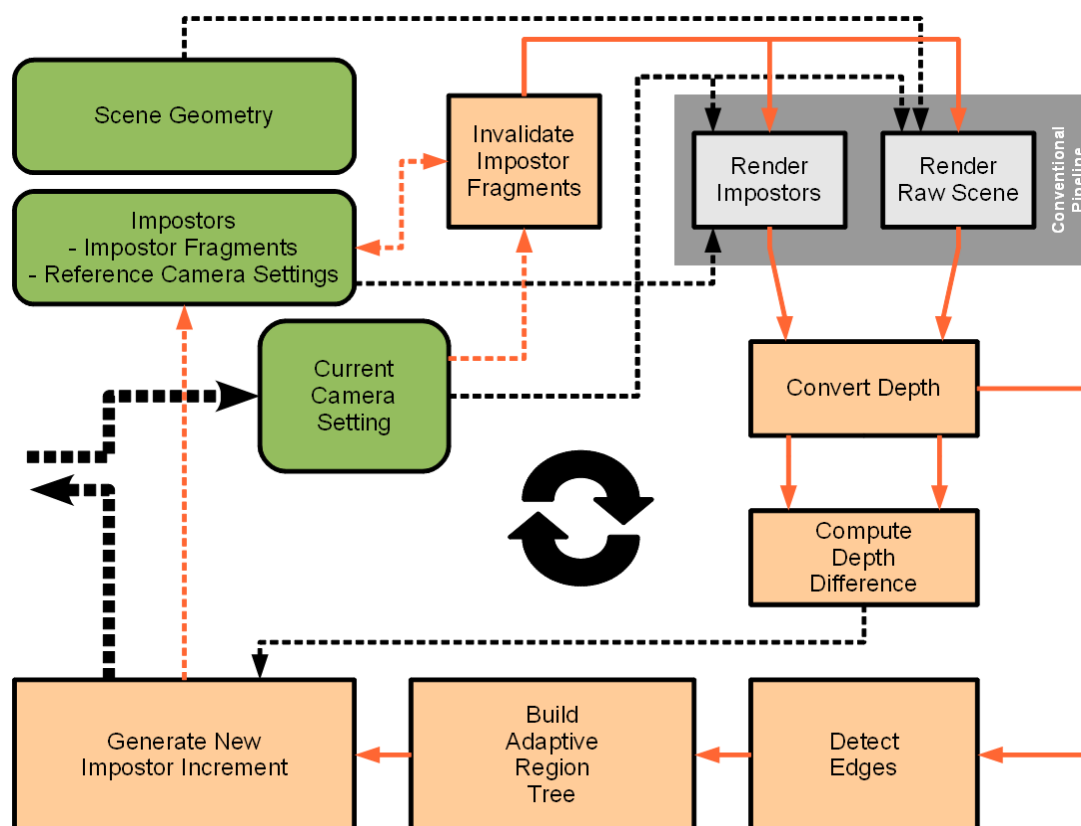


Figure 2: Incremental Impostor Streaming server core functionality

3.1 Impostor Reusability and Invalidation

Incremental Impostor Streaming reuses existing impostors, because client camera position and direction often do not change rapidly. In fact, frame to frame coherence solely depends on camera changes. To minimise network load, the server does not generate a full screen impostor but impostor increments which represent only those parts of the current view that have changed beyond a specific error measure.

Prior to impostor generation, existing impostors which no longer contribute to the new camera configuration are invalidated. There are three main causes that render an impostor increment invalid:

- 1) Resolution Mismatch (see Figure 3): As outlined before, an impostor mesh gets combined with the conventional pipeline's colour output at the reference position, because it is essentially a depth mesh based screen shot. Thus, its original resolution will be too high or too low depending on how much the camera moved closer towards or farther away from the impostor's position. This results in either blurriness or undersampling problems, respectively. At least in case of moving farther away, undersampling problems can be handled by generating mip-maps of the impostor's colour texture. In general, moving away results in a waste of texture memory.

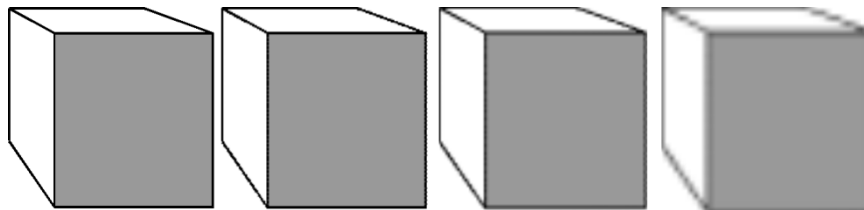


Figure 3: *Impostor resolution mismatch*

To cope with resolution mismatch, the relative Euclidean distance between the impostor's reference camera position and the current camera position must not exceed a given threshold. In this way, severity of resolution mismatch can effectively be bounded.

- 2) Skins (see Figure 4): Skins are “long” triangles that arise if an impostor mesh (a textured depth mesh – TDM) does not get disconnected properly at regions of high depth complexity. The IIS approach incorporates discontinuity detection on TDMs, but in some cases – especially at edges that are viewed from acute angles – depth differences between neighbouring pixels fall below the threshold, even if they represent non-connected scene components. Typically, skins look like the example shown in Figure 4: The upper box's marked edge cannot be detected completely, because the depth values of neighbouring pixels do not differ sufficiently. Section 3.3 describes the discontinuity detection algorithm at greater detail.

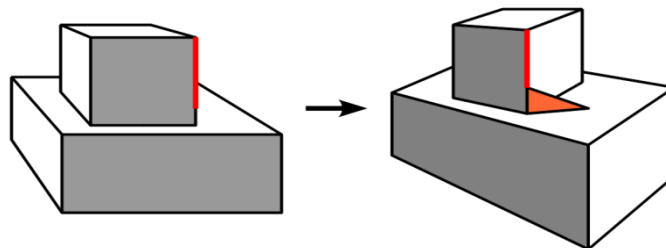


Figure 4: *Impostor skins due to limits of discontinuity detection*

Skins are avoided implicitly by thresholding the current camera's viewing angle deviation in relation to the reference camera viewing angle. In general, angles above 90° are infeasible for a correct impostor representation, because information about the backside of an observed scene part is never contained in an impostor. In practice, angle deviation should be limited to values of about 10° .

- 3) Disocclusion (see Figure 5): Disocclusion causes the most severe visibility errors. In fact, disocclusion is volitional – holes or cracks appear if a disconnected mesh gets viewed by a camera displaced from the reference position. Depending on camera movement or impostor fragment discontinuity, transparent regions of different size will appear, and it is not clear, whether these regions contain previously invisible geometry or not.

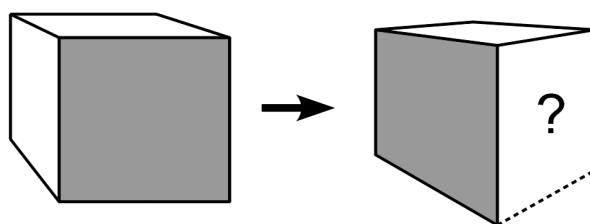


Figure 5: *Impostor Disocclusion: Camera movement uncovers previously invisible geometry*

Handling disocclusion is the most expensive part of impostor generation. As no information about occluded scene parts is available in any existing impostor, disocclusion is implicit and cannot be avoided. The next section describes how previously invisible scene parts are integrated into the current impostor database.

3.2 Handling Disocclusion through Depth Comparison

After applying distance and angle deviation metrics as explained in the last section, only valid impostor parts remain. To distinguish between disoccluded scene parts and scene parts that do not contain geometry, two data sets are rendered on the server from the current client camera configuration:

- Raw scene geometry (rendered conventionally)
- Current impostor database

Both rendering operations generate 3D pipeline output in form of colour and depth textures. To determine which parts of the current scene need to be included in a new impostor increment, the depth difference between the two depth textures is measured pixelwise. All pixels having a depth difference larger than a threshold are chosen to form a new impostor increment.

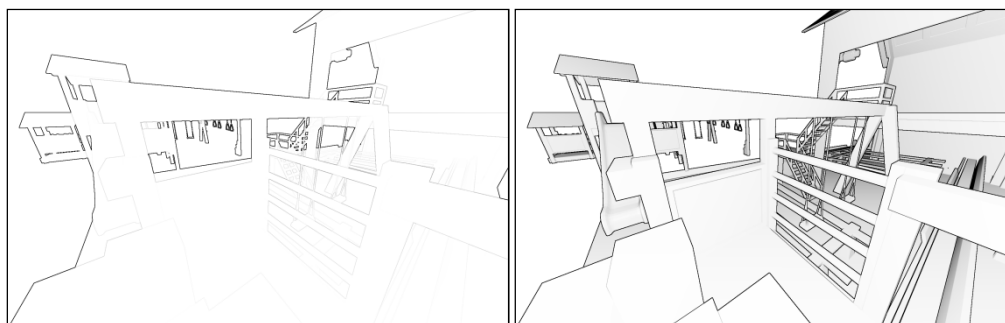
3.3 Depth Edge Detection and Discontinuities

Generating a full screen pixel-exact TDM is not feasible, because triangle count for a single impostor would be enormous. This means that the actual impostor triangle count needs to be minimised, while respecting an error threshold on depth deviation. Image space regions of higher depth complexity will be represented by more triangles while regions of lower depth complexity need fewer triangles. Optimising triangle count also yields an impostor representation that can be transferred and rendered much more efficiently.

To actually detect regions of high depth complexity, the Scharr operator is used [Sch00]. Like many other edge detection algorithms from image processing, the Scharr operator features a 3x3 convolution matrix that calculates the first derivative of the depth values. The main problem here is that edge detection cannot be applied directly to the raw depth texture. This is because conventional pipeline depth output is subject to extreme perspective warping and would yield severe contrast problems. Depending on the situation, raw depth edges are detected either in vicinity to the camera or far away, but not across the full scene depth range.

To cope with depth contrast problems, raw depth is at first linearised. Linear depth still does not suffice to correctly detect edges nearby and far away at the same time; in fact contrast problems are just as severe. The solution is to invert linear depth in a second processing step, resulting in a balanced depth edge amplification across the whole depth range. Figure 6 show example edge detection filter results generated using linear and inverse linear depth values.

A textured depth mesh (TDM) is usually connected, which is unfavourable in case of depth edges. The IIS approach maps the detected edges onto the original TDM and disconnects the mesh at the appropriate positions.



a) Linear depth (far focus)

b) Linear depth (near focus)



c) Inverse linear depth

Figure 6: Comparison of linear and inverse linear depth: Inverse linear depth results in high-contrast depth edge amplification.

3.4 Adaptive Mesh Generation

The last processing step resulted in a disconnected mesh and a corresponding depth edge image. To actually increase triangle count in image space regions of higher depth complexity, a space subdivision technique is used. Incremental Impostor Streaming utilises a special variant of a kD-Tree [Ben75] to subdivide image space; we call it Adaptive Region Tree. This tree is axis and pipeline output pixel aligned to allow for pixel-exact impostor mesh approximation, if needed. Figure 7 show an example ART (displayed as overlay mesh) applied to real pipeline output. One can see clearly, how regions of higher depth complexity result in more fine-grained image space subdivision. The most important subdivision condition is the existence of a depth edge (obtained by thresholding depth edge detection output). If there is a depth edge within the region cor-

responding to an ART node, the node is subdivided into two halves. Each ART node is subsequently mapped to a triangular representation which makes the final impostor mesh.

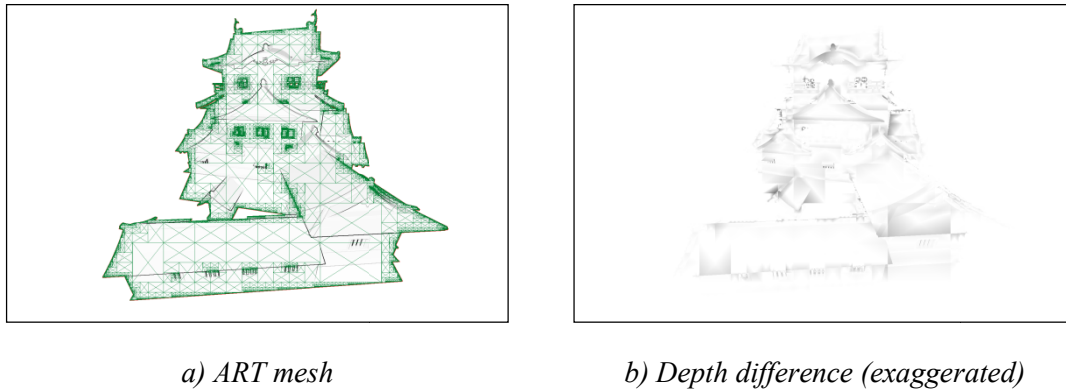


Figure 7: Example ART configuration on real pipeline output

3.5 TDM Generation, Transfer to the Client and Visualisation

To finalise impostor generation, all information generated in the previous steps is used to produce the wanted impostor increment.

The depth difference obtained in Section 3.2 determines which parts of the scene should be imaged by new impostors. After discontinuity detection using edge detection on the corresponding height map, an ART will be generated, which's nodes will be mapped to triangles. The resulting impostor mesh and its corresponding colour texture are then transferred to the client where they are to be displayed. Note that impostor display can be done by conventional means, i.e. the client does not need to know any impostor construction specifics. The client only has to render a conventional texture mapped on a triangular mesh, even if the rendering process on the server originally required advanced rendering capabilities.

4 Conclusions and Future Work

4.1 Accomplished Results

Incremental Impostor Streaming to visualise massive 3D scenes at remote clients has been prototypically implemented and tested. Some important advantages compared with conventional raw scene triangle rasterisation could be proved:

- 1) Client-side impostor rendering complexity is independent of scene size.
- 2) Rendering complexity depends on client screen resolution and preferred quality settings only. Quality settings include an edge detection threshold as outlined in Section 3 and a maximum ART depth. The maximum ART subdivision level di-

rectly influences the number of triangles generated for an impostor. To avoid exceeding the client-supplied triangle budget, ART subdivision can be stopped before reaching pixel level.

This means that requirements on client hardware capability can be arbitrarily low. Even lowest-end consumer graphics hardware is able to display a low-resolution texture mapped on a triangular mesh. The server automatically maintains mesh and texture resolution to suit the client's capabilities.

Assuming that the server would have to render the raw scene regularly anyway, Incremental Impostor Streaming only adds a small amount of processing load. In essence, only the time needed to construct an impostor is added. Since impostors are generated exclusively from raw pipeline output, no special server rendering requirements exist. A server-side standard 3D graphics pipeline generating a perspective depth buffer and corresponding colour data is the minimum requirement.

Certain IIS design decisions, for example the straightforward impostor generation based on pipeline output, allow for some of the algorithms used to be implemented completely on a GPU. Especially depth value conversion and depth edge detection can be accelerated by multiple orders of magnitude even if processed on a mid-performance GPU. Implementing these algorithms as GPU shaders in the proof-of-concept prototype sped up the process from 1.2 seconds to below a millisecond. With the advent of general purpose GPU computing, implementing most other parts of IIS on a GPU is expected to be feasible in the very near future.

With IIS, real-time rendering of massive 3D models is possible, even on low end systems. This is due to the fact that most rendering load is transferred to the server. The current impostor database state will never force the client to render impostors which exceed its current triangle budget. The server itself does not need to implement special pipeline functions since the IIS pipeline extension can be set up on top of a conventional pipeline easily. Because the client is able to quickly render the current impostor database, user interactivity and responsiveness is guaranteed at any time. Experience shows that especially responsiveness is far more important than immediate availability of disoccluded scene parts. A sudden camera turning, for example, results in visible impostor cracks or holes that are subsequently filled with the corresponding scene parts. Missing data is filled in asynchronously allowing the client to maintain a stable frame rate.

An example for camera turning is given in Figure 8. Due to the incremental nature of IIS, a very big part (around 70%) of existing impostors can be reused. Solid regions represent image space regions of very high depth difference. This information is used to create the actual impostor increment.

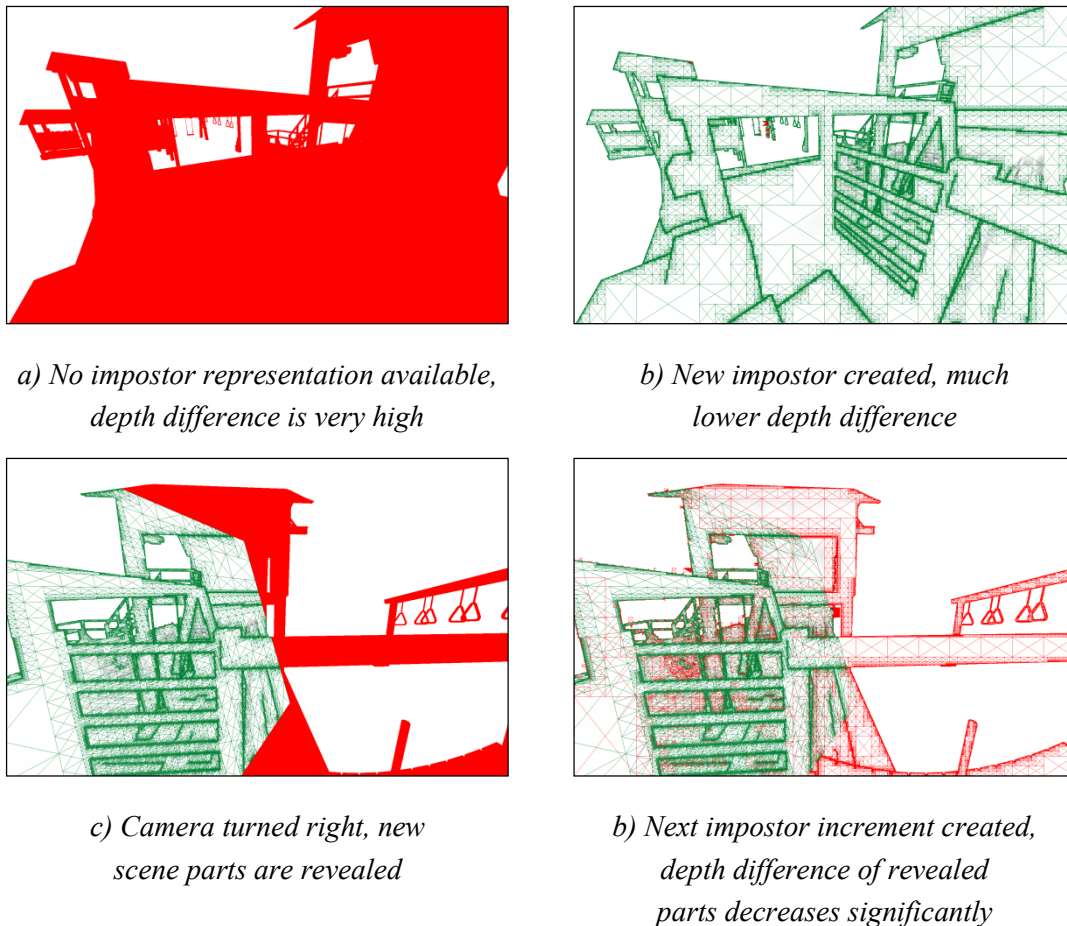


Figure 8: Camera Turning Example

To give a hint on typical client-side rendering load, a few measurements for the scene shown in Figure 8 were made. In the given example, a maximum ART depth of 22 (saturating a 1920x1200 viewport down to pixel level) resulted in about 75 000 triangles. An even lower amount of triangles, 5 400, was generated when capping the ART depth to 14. Even then, a negligibly low depth error has been obtained. Rendering such amounts of triangles is easily possible at real-time speeds, even on very low-end graphics hardware.

In the context of remote CAD model viewing or editing, another IIS property can be utilised: Raw scene data is never transferred to the client, because the client displays impostor geometry exclusively. This means that possible raw data security issues must be dealt with only at server-side.

The image-based nature of IIS imposes almost no additional constraints on the rendering techniques used. For example, lighting (forward as well as deferred methods) can be applied by generating and transferring an additional geometric normal texture. Because each impostor is a standard textured mesh, flexible GPU shading is possible both at server and client side.

In general, quality can be steered to any direction, depending on the situation. Higher network bandwidth and processing power boosts quality at any time by design. This means that IIS scales very well with the limitations imposed by the underlying network and client graphics hardware, which together constitute its two main bottlenecks.

4.2 Future Work

Future research will take many directions. To increase quality, especially for nearby objects, hybrid rendering strategies must be developed so that some parts of the scene geometry are rendered conventionally. This is expected to increase rendering complexity by a constant amount only. At the moment, IIS incorporates no specific method to cope with animated geometry. Rendering conventional geometry is an obvious, but possibly inefficient solution to this. As with many rendering systems, transparency is a problem that must be solved also for IIS rendering.

To utilise additional client processing power, a lot of work could be transferred to the client. For example, lighting computations could be done by the client. In this way, only basic lighting information (normals, light positions, light types etc.) has to be transferred to the client. This would also unburden the server, because TDM generation could in this case be done using raw scene rendering without special effects.

Other possible client-side extensions include impostor database optimisation. To further decrease impostor triangle count at the client side, various culling techniques could be used. This allows for a faster reaction to changing triangle budget and bandwidth settings. Adding Quality-of-Service to IIS impostor transfers is another field, which could be addressed by incorporating state-of-the-art mesh and colour compression techniques.

In general, IIS provides a flexible, image-based remote rendering framework with high scalability. Due to decoupling of raw scene rendering load, impostor generation and corresponding client-side impostor rendering, it will be possible to integrate other techniques at known costs.

5 References

- [Ben75] BENTLEY, J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM* 18(9):509-517, 1975
- [DSS+99] DÉCORET, X.; SCHAUFLER, G.; SILLION, F.; DORSEY, J.: Multi-layered impostors for accelerated rendering. *Computer Graphics Forum* 18(3):61-73, 1999
- [HPB05] HEYER, M.; PFÜTZNER, S.; BRÜDERLIN, B.: Visualization server for very large Virtual Reality scenes. *Proceedings of 4. Paderborner Workshop "Augmented & Virtual Reality in der Produktentstehung"*, Paderborn, 2005
- [Jes05] JESCHKE, S.: Accelerating the rendering process using impostors. PhD Thesis, University of Rostock, 2005
- [Sch00] SCHARR, H.: Optimal Operators in Digital Image Processing, PhD Thesis, University of Heidelberg, 2000

- [WM03] WILSON, A.; MANOCHA, D.: Simplifying complex environments using incremental textured depth meshes. *ACM Transactions on Graphics* 22(3):678-688, 2003

Authors

Dipl.-Inf. Johannes Ghiletiuc graduated from Technical University of Ilmenau in 2010. His Diploma thesis in Computer Science carries the title “Accelerating Real-time Rendering with Image-based Impostors”. He intends to pursue an academic career in the field of computer graphics. His research interests include advanced real-time rendering techniques, image-based rendering approaches and high-profile GPU programming.

Dipl.-Inf. Markus Färber holds a Computer Science Diploma degree from Dresden University of Technology and is currently a member of the Computer Graphics Group at the Technical University of Ilmenau. He is the scientific and administrative coordinator of the Graduate School on Image Processing and Image Interpretation. His research interests focus on constraint-based geometric modelling, human computer interaction techniques and fundamental algorithms for computer graphics problems.

Prof. Dr. Beat Brüderlin is professor for computer graphics and chair of the Computer Graphics group at the Technical University of Ilmenau. After his PhD in Computer Science from the Swiss Federal Institute of Technology Zurich in 1987 he taught computer graphics and geometric modelling at the Department of Computer Science at the University of Utah. Professor Brüderlin is the head of the Steinbeis-Transferzentrum für Interaktive Computergrafiksysteme/CAD. In 2004 he initiated formation of 3Dinteractive, a software development company which supplies major aircraft and automotive industries with software for visualisation and editing of massive 3D data sets. His research interests include advanced rendering techniques, interactive conceptual modelling and geometric constraint solving.