Adapting Ray Tracing to Spatial Augmented Reality

Markus Broecker
Wearable Computer Lab
University of South Australia
markus.broecker@unisa.edu.au

Bruce H. Thomas
Wearable Computer Lab
University of South Australia
bruce.thomas@unisa.edu.au

Ross T. Smith
Wearable Computer Lab
University of South Australia
ross.t.smith@unisa.edu.au

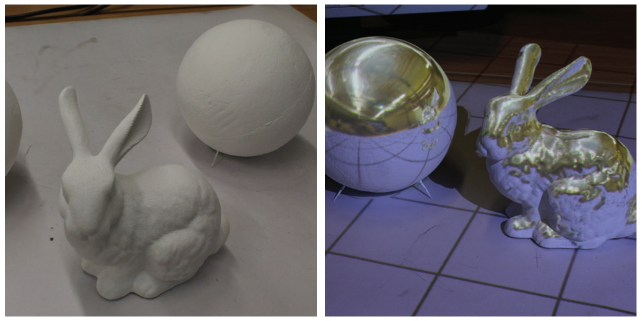

Figure 1: Non-augmented props and view-dependent ray tracing using a Spatial AR system.

Extended Abstracts of the
IEEE International Symposium on Mixed and Augmented Reality 2013
Science and Technology Proceedings
1 - 4 October 2013, Adelaide, SA, Australia
978-1-4799-2869-9/13/$31.00 ©2013 IEEE

Abstract

Ray tracing is an elegant and intuitive image generation method. The introduction of GPU-accelerated ray tracing and corresponding software frameworks makes this rendering technique a viable option for Augmented Reality applications. Spatial Augmented Reality employs projectors to illuminate physical models and is used in fields that require photorealism, such as design and prototyping. Ray tracing can be used to great effect in this Augmented Reality environment to create scenes of high visual fidelity. However, the peculiarities of SAR systems require that core ray tracing algorithms be adapted to this new rendering environment. This paper highlights the problems involved in using ray tracing in a SAR environment and provides solutions to overcome them. In particular, the following issues are addressed: ray generation, hybrid rendering and view-dependent rendering.

Author Keywords

Ray tracing, Augmented Reality, Spatial AR

ACM Classification Keywords

H.5.1 [Multimedia Information Systems]: Artificial, augmented and virtual realities; I.3.7 [Computer Graphics]: Raytracing,colour, shading, shadowing and texture

Introduction

Ray tracing is an image generation method that enables the creation of high-quality computer generated images. Its recursive nature naturally allows for complex graphics effects, such as shadows and reflections [3, 9]. The shading of a fragment in ray tracing can take into account the whole scene thus allowing the implementation of global illumination methods as well as realistic reflections, refractions or shadows. These methods are possible in rasterisation but require complex multi-step algorithms to achieve a similar result.

Ray tracing offers other advantages over rasterisation as well. Firstly, ray tracing allows for a richer set of arbitrary data input and output operations and complex shading methods. Secondly, it is able to perform physically-based rendering; thirdly, it offers very high quality images compared to rasterised graphics – as an example, path tracing [4] methods are often used to create reference images to which other graphics algorithms are compared.

Until recently, ray tracing in Augmented Reality (AR) was not a viable option, as it is a slower image generation method compared to hardware accelerated rasterisation. One fundamental requirement of AR graphics is that it has to be capable of real-time, interactive performance [1]. The advent of GPGPU[1] and supporting software, such as CUDA[2] and OptiX [7], allows the use of powerful, programmable and parallel-processing capable graphics cards to accelerate ray tracing to real-time frame rates.

Spatial Augmented Reality (SAR) is a specialised form of AR which uses projectors as the primary display device [2]. Physical models (called props, usually painted white) are illuminated by these projectors, allowing the system to simulate different surface properties (see the left and right images in Figure 1). The image on the left of Figure 1 depicts examples of SAR props lit in environment light while the right image shows the same props simulating a reflecting metallic surface. SAR offers an intuitive approach to AR, without requiring the user to wear a head-mounted display or hold a hand-held device. As users do not have to handle the display device, the SAR approach scales well to a group of users [6], compared to traditional see-through AR methods. It also offers affordance with the physical presence of props, further increasing the realism and perception of the virtual display.

In this paper we present a set of ray tracing algorithms designed to operate under SAR conditions. SAR systems have numerous peculiarities that can result in image artefacts and, in the worst case, wrong images, if these ray tracing algorithms are implemented naïvely. In particular, three core techniques are the focus of this work. First, ray generation based on arbitrary matrices will be investigated. Secondly, hybrid rendering allowing the combination rasterisation and ray tracing will be presented. Finally, view-dependent rendering techniques, based on modification of first-order intersection results are described.

These three core techniques build the foundation for further ray tracing methods and algorithms.

Ray Generation

Each projector in a SAR system is defined by a projection and a view matrix, which are calculated by an external calibration program based on camera-pose estimation. These matrices are approximations of projection parameters (calculated using camera pose estimation) and

[1]General-purpose computing on the GPU.
[2]nvidia.com/object/cuda

$$M_s = \begin{pmatrix} \frac{d}{a} & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$M_o = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{n+f}{n-f} & \frac{2nf}{n-f} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Figure 2: Projection matrices.

Projectors have a fixed resolution. *Local resolution* describes the actual resolution of a projector on a given surface area of the projection surface. Distance between projector and surface as well as the relative shape and angle of the surface to the projector influence the local resolution.

are not created from direct measurements of the projection parameters. Figure 2 shows two projection matrices for rasterised graphics pipelines.

Matrix M_s is the symmetric frustum projection matrix, as it is commonly used and created by many to standard library functions[3]. Matrix M_o shows a more general, oblique projection matrix [8] that defines an asymmetrical frustum. Intrinsic parameters, which describe projection parameters such as focal length, can be recovered from projection matrices. These values can be used to generate primary rays in ray tracing; however, in the case of SAR, they must only be interpreted as approximations. Errors in the recovery of these parameters will cause the resulting image to diverge from the rasterised OpenGL image created by the rest of the system. Even small numerical errors are multiplied by long projection distances. For example, the projectors employed in our system have a resolution of 1280×800, which yields an image width-to-height ration of 1.6. Parameter recovery calculated the aspect ratio for two projectors with individual calibrations as 1.59 and 1.61.

Our solution to create rays in the projector's frustum is to calculate the frustum's eight defining corner vertices in world-space. Once calculated, the primary rays can be created through bilinear interpolation of the frustum's near and far plane. The frustum is calculated in world-space by the transformation pipeline used in rasterisation graphics (see Equations 1 – 3). The eight coordinates of the clip-space cube (p_{clip}) are transformed by the inverted projector matrix ($M^{-1}_{projector}$). $M_{projector}$ maps points from world- to clip-space, so the inverse transformation $M^{-1}_{projector}$ maps from clip to world space. The transformed coordinates need to be divided by the respective homogenous coordinates (Equation 3) after transformation. p'_{world} describes the eight corner vertices of the frustum in world coordinates.

[3]For example by calling `gluPerspective`.

$$M_{projector} = M_{proj} \cdot P_{view}, \tag{1}$$
$$p_{world} = M^{-1}_{projector} \cdot p_{clip}, \tag{2}$$
$$p_{clip} \in \left(\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 1 \\ 1 \end{pmatrix} \cdots \right),$$
$$p'_{world} = p_{world} / p_{world}.w. \tag{3}$$

Figure 3 shows the interpolated points on the frustum's near and far plane. A ray R is defined by creating a start point X and end point Y on the frustum's near and far planes.

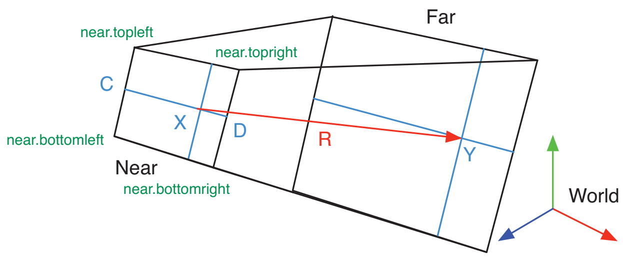

Figure 3: A projector's frustum in world coordinates.

The low local resolution of projectors illuminating props results in very noticeable projection artefacts due to high-frequency texturing (as well as reflections and refractions, as seen in Figure 4). Multi-sampling techniques are therefore required to provide higher image quality (see Figure 5). Jittered grid multi-sampling was

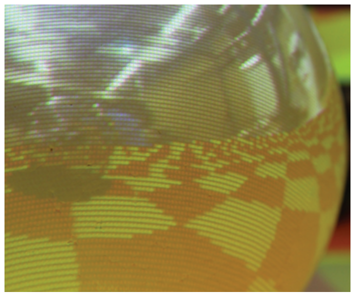

Figure 4: Strong aliasing due to low local resolution and spread of secondary rays.

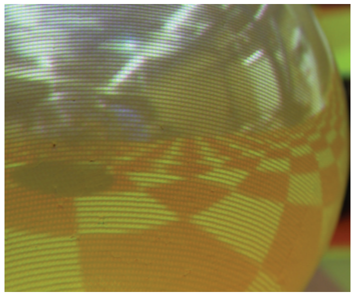

Figure 5: Multi-sampling is required in a SAR system and improves display quality.

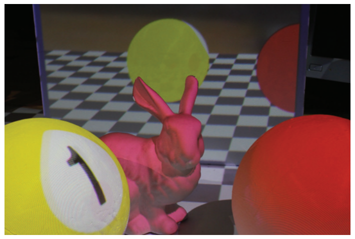

Figure 6: An example of hybrid rendering. Notice the correct occlusion of spheres and physical bunny and the missing reflection of the bunny in the mirror.

chosen for our implementation. Random numbers, required for multi-sampling, are generated on the fly on the GPU using a pseudo random number generator with a small common state buffer.

Hybrid Rendering

In our specific case, we wanted to integrate the ray tracer into an existing rasterisation based SAR system. The output of the ray tracer should therefore combine correctly – that is, provide correct occlusion – with existing graphics, whether they are rendered before or after the ray tracing step. Hybrid rendering, in our case, refers to combining rasterisation and ray tracing to generate images, not accelerating ray tracing using rasterisation.

Ray tracing implicitly provides the linear intersection length along its rays. However, the depth buffer commonly used in rasterisation graphics is not linear, but follows a $\frac{1}{z}$ curve. Linear depth coordinates from the ray tracer must be converted to the non-linear depth buffer space so that the hardware is able to resolve occlusions correctly between ray tracing and rasterisation graphics.

One solution is to calculate the new depth coordinates in a fragment shader, while the screen-filling quad with the ray tracing result is written to the back buffer. The algorithm starts with determining the world position of the current intersection. To do so, we perform the same ray casting in the fragment shader for each fragment, as described for the ray generation process. Each projector frustum's parameters are stored in a read-only buffer which is passed to the shader and the linear distance along each ray is stored in the alpha channel of the texture.

Calculating the depth follows the standard rasterisation pipeline after the world position of the intersection point has been calculated. The point in world coordinates gets transformed into normalised device coordinates by the projector's transformation matrices and the depth divide. Clipping need not be performed, as all points for transformation are known to be in the current projector's field of view. The last step transforms the normalised device coordinates into window coordinates – that is, the depth values are scaled to lie in $[0 \cdots 1]$. The z value of the transformed vertex is written to the depth buffer for each fragment while the stored ray traced colour value is written to the back buffer. The following equations show the steps for this algorithm:

$$
\begin{aligned}
p_{world} &= ray_{origin} + ray_{direction} \cdot distance, && (4)\\
p_{clip} &= P_{proj} \cdot P_{view} \cdot p_{world}, && (5)\\
p_{NDC} &= p_{clip}/p_{clip}.w, && (6)\\
z &= p_{NDC}.z * 0.5 + 0.5. && (7)
\end{aligned}
$$

Figure 6 shows the correct occlusion results between multiple ray traced and a rasterised object. Notice that only geometry that is ray traced (in this case the spheres and the ground) will be reflected and reflect other ray traced geometry – rasterised geometry (the physical Stanford bunny) is absent in the ray traced reflection.

Multisampling averages all pixel samples in one pixel – including the linear depth. This leads to incorrect depth results, especially along the edges of an object, where some rays within that pixel miss the object and have a distinctly larger intersection depth than the other rays which hit the object. To enable correct blending, we only use the closest intersection distance of all sample rays, which we store as the depth value.

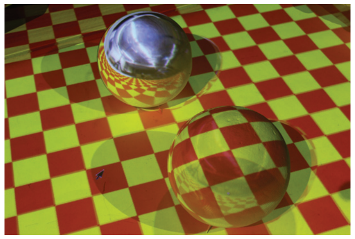

Figure 7: The scene from Whitted's ray tracing paper reimplemented in SAR.

Primary order rays and intersections are cast from the projector and intersect an object. They create *secondary* rays and intersections through reflective or transparent materials.

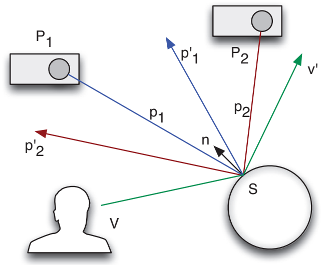

Figure 8: Two projectors – P_1 and P_2 illuminate a prop S with the surface normal $\mathbf{n}$. The viewing positions V must be taken into account when creating secondary rays to create the correct reflection or refraction.

View-dependent Rendering

A SAR system has no inherent knowledge of the position of the viewer. During rendering, each projector becomes the current active viewer and renders the world from its position. However, many effects, such as specular highlights, reflections or refractions, require knowledge of the user's view position. This creates a class of techniques called 'view-dependent rendering', in which the user's viewpoint is tracked and rendering takes this new information into account.

Recursive ray tracing enabled a direct implementation of view-dependent rendering. Ray intersections can create new rays, changing their direction and position depending on material properties and the user's position. Reflections can be calculated accurately instead of relying on texture effects. This allows for view-dependent effects on geometry of any shape, avoids the problems of aliasing and employs the recursive nature of the ray tracer efficiently. Figure 7 shows a reimplementation of the scene found in Whitted's ray tracing paper [9]. The sphere in the background shows reflection of near geometry (through ray tracing) and far geometry (through environment mapping). The sphere in the foreground shows refraction.

It is important to differentiate between primary and secondary rays in view-dependent rendering techniques. Primary rays are cast from the projector into the scene and primary hits describe intersections of these rays with the geometry of the scene. Secondary rays are created by reflecting or refracting primary or secondary rays on the objects, at the point of intersection and based on material properties (for example, reflectivity or fresnel number).

Figure 8 highlights the difference between primary and secondary rays when using ray tracing in SAR. Two projectors P_1 and P_2 illuminate a physical prop S at a single pixel with the normal $\mathbf{n}$ at that point. A tracked viewer V is observing the scene. When implementing reflections without regard to the viewing position, the primary rays $\mathbf{p_1}$ and $\mathbf{p_2}$ are reflected at the normal to create the secondary rays $\mathbf{p_1}'$ and $\mathbf{p_2}'$ respectively. However, it yields the wrong reflection result for the viewer V. This effect is exacerbated when multiple projectors are illuminating the same pixel and casting different reflection rays on that point. If, however, the secondary ray is created by reflecting the incoming view vector, the correct result is achieved – independent of any projector's position.

Conclusion

This paper describes ray tracing in the context of SAR. Projector calibration in SAR provides us with view and projection matrices through pose estimation. Decomposing these matrices to calculate intrinsic parameters might lead to incorrect projections. We therefore implemented ray casting based on the frustum's shape that these matrices describe. This results in images that have the same geometric transformations as the rasterisation pipeline describes; thus, we can combine ray traced and rasterised images. To enable correct occlusion results, the linear intersection distance along a ray has to be converted to non-linear z-buffer space. The previously calculated matrices were used to assist, and this was implemented in a shader and each fragment's depth is processed while the ray tracing result is written to the screen.

One of the key differences between ray tracing with a traditional display and in the SAR context is the clear distinction between primary and secondary rays. In traditional ray tracers, no distinction needs to be made;

however, SAR requires this for the calculation of reflections, refractions and also miss rays. Primary rays are always cast from the projector in SAR and are used to create initial intersections. Secondary rays include further intersections and reflections.

Finally, we note that with the available software libraries and interfaces for ray tracing on the GPU, the entry barrier to implement efficient, specialised ray tracing is low: the available computing power in modern graphics cards enables ray tracing in real-time.

There are many different uses for ray tracing in a SAR environment. High-quality simulation of surface materials (for example, car paint) in a design-oriented SAR application is one use case, which is also independently researched by other authors [5]. Simulation of participating media (for example, liquids) and rendering of volume data sets is another. Finally, some CAD models represented by freeform surfaces, such as NURBS, can be evaluated directly using ray tracing without triangulation. This would allow a more direct workflow in a design application of SAR.

Acknowledgements

We would like to acknowledge the work of Michael Marner in our SAR software and hardware infrastructure. We would also like to thank members of our Wearable Computer Lab, Erin Zimmerman and Sandra Bennett for proof-reading the paper.

References

[1] Azuma, R. T. A Survey of Augmented Reality. *Presence 6* (August 1997), 355–385.

[2] Bimber, O., and Raskar, R. *Spatial Augmented Reality - Merging Real and Virtual Worlds*. A K Peters, Ltd., 2005.

[3] Cook, R., Porter, T., and Carpenter, L. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, vol. 18, ACM (1984), 137–145.

[4] Lafortune, E. P., and Willems, Y. D. Bi-directional path tracing. In *Proceedings of CompuGraphics*, vol. 93 (1993), 145–153.

[5] Menk, C., Jundt, E., and Koch, R. Visualisation techniques for using spatial augmented reality in the design process of a car. *Comput. Graph. Forum 30*, 8 (2011), 2354–2366.

[6] Mine, M., van Baar, J., Grundhofer, A., Rose, D., and Yang, B. Projection-based augmented reality in disney theme parks. *Computer 45*, 7 (july 2012), 32 –40.

[7] Parker, S., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., et al. Optix: A general purpose ray tracing engine. *ACM Transactions on Graphics (TOG) 29*, 4 (2010), 66.

[8] Verth, J. V., and Bishop, L. *Essential Mathematics for Games and Interactive Applications: A Programmer's Guide*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[9] Whitted, T. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (1980), 343–349.