Fast Gather-based Construction of Stereoscopic Images Using Reprojection

Marries van de Hoef, Bas Zalmstra

May 5, 2011
Utrecht University

Abstract

We developed a very fast reprojection technique to generate stereoscopic images from a 2D image with depth information. The technique is gather-based and therefore very fast on current graphics hardware. The depth information is sampled at a specific offset which provides the depth to reproject from the left or right camera to the center camera.

We provide benchmarks and a quality analysis which demonstrate a very fast execution time and a good image quality while using a moderate stereoscopic strength.

1 Introduction

The popularity of stereoscopic 3D for entertainment purposes is growing rapidly. Stereoscopic 3D is common in movie theatres and it is moving towards the living room due to the increasing support of stereoscopy on consumer displays. These developments have boosted the demand for stereoscopic 3D support in games. Steroscopy does not only increase the immersion in games, but the depth perception can also improve the gameplay.

Generating stereoscopic images in games requires two images to be rendered; one for the left eye and one for the right eye. This doubles the processing power required. Most games maximize their hardware usage to be able to render the most aesthetic images possible. When two images have to be rendered instead of one, the image quality has to be reduced drastically by rendering at a lower resolution and with less detail.

An alternative for the rendering-twice approach is to reproject a single image to two images, based on depth information of the single image. The similarity between the images for the left and right eye is exploited in this reprojection approach. To use the same rendering settings as the 2D approach and thus retaining the 2D aesthetics, the reprojection has to be performed as fast as possible. The quality of the reprojected images has to approach the quality of independently rendered images to be useful in practice.
We developed a technique which combines a very fast execution time with a good reprojection quality. This technique restricts the eye separation to moderate values. A higher eye separation reduces the reprojection quality. We extended the technique to produce a relatively good reprojection quality at a high eye separation, but this extension requires a longer execution time.

1.1 Related work

Crytek announced the ScreenSpace Reprojection Stereo technique [1] at the Game Developers Conference Europe 2010 in Cologne. The technique was not explained, only the characteristics of the technique were revealed in the following slide:

- Fully gather-based, no warped grid or point splatting
  - Works in single-pass pixel shader
- Reproject pixel into space of left/right eye cameras
  - Done in screen space by computing offset based on pixel depth and stereo parameters
- Resample backbuffer with bilinear filtering

The technique we developed shares most characteristics. We do not know if our implementation is equal to Crytek’s because no further information is publicly available on Crytek’s implementation.

2 Reprojection Analysis

2.1 Reprojection introduction

Three different cameras are used in reprojection. The center camera is used to render the scene with depth information. The center camera image is reprojected to the left and right camera using the depth information. The left and right camera images are the resulting stereoscopic images.

Reprojection can appear very straightforward at first. The pixels of the center camera image are reprojected to the left or right image using the following steps (visualized in Figure 1):

- The current pixel is transformed from screen space to view space using the inverse projection matrix. This results in a ray.


---

1Using three cameras is favored over a two camera approach. In a two camera approach, the left camera is used to render the scene and the resulting left camera image is reprojected to the right camera. The reprojection distance in the two camera approach is larger than the reprojection distance in the three camera approach, which decreases the image quality.
Figure 1: Reprojecting a point from the center image to the left eye image. From left to right: The point on the center image, the backprojected ray from the center camera, finding the position on the ray using depth information, the position in world space, projecting the point onto the left eye image.

- The original position of the pixel lies on the ray. The depth of the original point is equal to the stored depth information of this pixel. The depth information is used to find the original point on the ray. The original point is in the view space of the center camera.

- The point is translated from the center camera view space to the left camera view space.

- The point is projected to the left camera image using the off-axis projection matrix of the left camera.

The result is that for the same position in the world, the corresponding pixel in both the left and center camera image is known. The pixel in the left image should have the color of the corresponding pixel in the center image.

This approach is a point splatting approach: for each pixel, the final location in the left camera image is not known beforehand. Point splatting is not efficient on the current graphics hardware, it is designed for a gather-based approach.

### 2.2 Gather-based reprojection

In a gather-based approach, the color has to be calculated for a predetermined position. In this case, the predetermined position is one of the pixels in the image for the left or right eye camera. The reprojection technique can be made gather-based by inverting the approach discussed in 2.1. Instead of starting at a pixel in the center image and finding the corresponding pixel in the left image, we start in the left image and find the corresponding pixel in the center image. Figure 2 demonstrates this inverse approach. The major problem with this inversion is that the depth along the ray from the left image is unknown. In the center image, we had access to the corresponding depth information because the scene was rendered from the perspective of the center camera. The main issue resolved in this paper is how the depth information of the left image can be recovered from the depth information of the center image.
2.3 Depth retrieval

The depth information for the left camera is derived from the depth information of the center camera. The depth information of the center camera is stored in a depth image where the depth is stored for each pixel. Figure 3 shows a top-down visualization of the depth information for each pixel in one row. This depth information can be regarded as a heightmap of the scene. The left camera looks at the same scene but from a different angle. The depth of a pixel in the left image is equal to the depth at which the corresponding ray intersects the heightmap of the scene. The intersection depth of the ray with the heightmap is found using ray marching. Figure 4 shows the left camera with a ray \( r \) along which the depth \( d_{ray} \) has to be retrieved. The ray march iteratively checks if an intersection with the heightmap has occurred for subsequent samples using the following steps:

- Select a position along the ray to sample for an intersection. The depth of this sampling position is \( d_{ray} \).
- Retrieve the depth \( d_{image} \) of the heightmap (depth image) for this sampling position by sampling the depth image at the corresponding image position. The image position is acquired by projecting the sample position to the center image.
- If \( d_{ray} \) is larger than \( d_{image} \), an intersection has occurred between this sample position and the previous position. If not, continue the ray march by selecting a new sample.

After the ray march terminates, the intersection position is known to be between the last sample and the second last sample. There are several options for approximating the final intersection position. The most simple approximation is to take the depth on the ray of the last sample. Another approximation method is explained in Section 4.2. The accuracy of the resulting intersection position depends on the density of the samples taken in the ray march and the approximation method.
In the parallax occlusion mapping technique, a very similar approach is used to find the intersection of a ray with a heightmap [2].

3 Performance Optimizations

3.1 Horizontal shift

The position of the left and right camera is obtained by moving along the x-axis of the center camera. The resulting shift in view space is always horizontal. Thus each reprojected pixel is shifted only horizontally. Because there is no vertical change, the y-coordinate of a pixel in the center camera image is the same as the y-coordinate of the corresponding pixel in the left camera image. Because the reprojected y-coordinate does not change, it can be omitted from all calculations.

When projecting a point on the ray to obtain the screen space coordinate, only the x-coordinate is relevant. The full matrix multiplication is replaced by only one multiplication and the perspective division.

3.2 Vertex shader

The reprojection is executed in a full screen pass. The vertex shader is executed for the four vertices. The output of the vertex shader is linearly interpolated in the rasterizer, and those interpolated values are used by the pixel shader. The pixel shader is executed for each pixel. At a resolution of 1280 × 720, the pixel shader is executed 921,600 times as opposed to the 4 executions of the vertex shader. Moving calculations from the pixel shader to the vertex shader results in a huge performance gain. Only linearly interpolatable calculations can be moved to the vertex shader because the rasterizer does only linear interpolation before the values are passed to the pixel shader.
The calculation of the direction of the ray can be moved entirely to the vertex shader. The multiplication with the inverse off-axis projection matrix is linear. The division by the w-term is also linear because the w-term is constant. The w-term after a multiplication with the inverse off-axis projection matrix only depends on the z- and w-term of the input. These terms (from the input vertices) are constant.

The calculation of the sample position along the ray and its projection can also be done in the vertex shader. The perspective division is linearly interpolatable because the w-term is constant. The w-term only depends on the z-term (depth) of the input position, which is constant for one sample.

The offset between the originating pixel in the left camera image and the sampling pixel in the center camera image only depends on the depth of the sample along the ray and the projection parameters. The projection parameters are constant and the depth of a sample is the same for all pixels across the screen. Therefore, the offset for a sample at a specific depth is constant for all pixels. These offsets can be pre-calculated, reducing the computations even further.

### 3.3 Sampling distribution

The distribution of the samples over the ray should not be uniform; instead it should increase with the depth. Figure 5 shows an increasing distribution, as opposed to the uniform distribution in Figure 4. If the sampling pattern is uniform, the distance between the projected positions of the samples decreases when the depth increases. This is caused by the vanishing point effect of perspective projection. A small distance between the projected positions results in oversampling and an unnecessary increase in execution time.

The depths of the samples can be defined by an exponential function. A different method is to select the pixel offset directly, which requires the calculation of the corresponding depth on the ray. The optimal distribution of the samples depends on the application and the required depth accuracy.

### 4 Quality Improvements

#### 4.1 Missing information

An inherent problem of reprojection is the missing information. Because the position of the camera is shifted, occluded areas of the scene become visible. No information is available on these areas because it was not stored in the render
Figure 6: Information is missing when reprojecting an image. The dashed lines show the parts of the scene which are not visible by the center camera, so this information is not available in the reprojection.

from the center camera. Figure 6 shows an example. The missing information cannot be recovered without addition information. The areas with missing information cause artifacts in the final image, so the size of the areas should be reduced and the artifacts should be as comfortable as possible.

The size of the areas with missing information is reduced by reducing the eye separation. But the stereoscopic effect is also reduced when reducing the eye separation. This is a trade-off between the intensity of the stereoscopic effect and the size of the areas with missing information. Viewing comfort is another reason to keep the eye separation moderate [3]. A moderate eye separation prevents eye strain, and enables the viewer to fuse the stereoscopic images easily. The ability to fuse the images also depends on the distance of the focal plane. The focal plane should be near because that is the natural focal distance when the user is watching the screen. With a close focal plane, the disparity of the stereoscopic images quickly increases with an increase of the eye separation. This can inhibit the fusion of the images by the viewer. To limit the disparity while retaining the near focal plane, the eye separation should be moderate.

Objects which are very close to the camera have larger areas with missing information, so this situation should be avoided. It is important to prevent objects from getting too close to the camera with stereoscopy in general because they have a large disparity.

The area with missing information causes different types of artifacts. The type of artifact depends on the depth of the intersection point found in the ray march. If the depth is lower than the actual depth of the intersection point, the area will be filled with color data from the background. If the depth is higher than the actual depth, the area will be filled with color data from the foreground object. Figure 7 shows the artifact when the area is filled with background data. This is the least noticeable in most cases and gives more symmetrical results across both eyes [4]. In the ray march, the background color data can be selected
Figure 7: The area with missing information is filled with background data which causes a repeat.

Another type of artifact is the lining artifact shown in Figure 8. This occurs when the area with missing information has multiple layers of depth with each their own transition from background to foreground. This artifact is resolved by adding a bias to the depth. The final depth should be lowered slightly to ensure that background color data is selected.

4.2 Intersection point approximation

The intersection in the ray march is found by comparing the depth on the ray $d_{ray}$ to the depth from the depth image $d_{image}$. The intersection point can be approximated in several ways. The sample point on the ray can be regarded as the intersection point. The screen space location of this point is already known because the depth image has been sampled there. The color image can be sampled at the same point to obtain the final color.

The intersection point can also be approximated using $d_{image}$. The intersection point is considered to be the position on the ray at depth $d_{image}$. This new point has to be projected to find the screen space point. The color image is sampled at the screen space point to retrieve the final color. The advantage of using $d_{image}$ to approximate the intersection point is that $d_{image}$ has much more detail. $d_{image}$ varies with the depth of the scene while $d_{ray}$ only varies when the number of used samples on the ray changes. Figure 9 shows the huge increase in depth detail when using $d_{image}$ instead of $d_{ray}$ to approximate the intersection point.
4.3 Color image sampling

An inherent problem of resampling the color image is that the exact color at a non-integer location cannot be retrieved. The image is filtered using nearest-neighbor filtering or bilinear filtering:

Nearest-neighbor filtering results in a sharp image, but the sampling location is not exact. The sampling location error is at most half a pixel.

Bilinear filtering uses exact sampling locations, which results in a better approximation of the correct color. The disadvantage is that the interpolation blurs the image.

5 Implementation

5.1 Implementation considerations

5.1.1 Edges of the screen

The color and depth images are being sampled outside its range at the left or right edge of the screen. This can be resolved by rendering the images at a higher horizontal resolution and reprojecting at the normal resolution.

Instead of solving the problem, the noticeability can also be reduced by mirroring the sampling location at the left and right side of the screen.
5.1.2 Transparency

Transparent geometry cannot be dealt with correctly. The depth image can store only one value at every pixel, so it cannot deal with multiple layers of visible geometry. The depth image should store the depth of either the opaque background or the transparent foreground. The other surface will have an incorrect depth.

This limitation can be circumvented by rendering transparent surfaces separately. These separate color and depth buffers for transparent geometry can be used to perform another reprojection pass. The reprojected transparent and opaque images have to be merged for each eye to obtain the final reprojected image.

5.1.3 Hybrid approach

Distortions caused by the reprojection can be removed completely by using a hybrid approach. The parts of the scene which are sensitive to distortions can be rendered twice on top of the reprojected images. This can be used to render transparent surfaces and near objects without distortions.

5.2 Algorithm

The algorithm presented in this section focuses on a fast performance while retaining a good quality when used with moderate stereoscopic parameters.

To achieve a very fast performance, the ray march is omitted. Instead, the approximation technique as explained in Section 4.2 is used directly. The approximation samples the depth image at the projected position of the sample on the ray. The depth of the single sample on the ray is chosen to be high, because this causes the area with missing information to be filled with background data. Algorithm 1 shows the steps in the algorithm.

For flat surfaces, this algorithm yields perfect results. The location in the depth image at which the depth is retrieved, can be slightly off because the ray march is omitted. But if the surface is flat, the retrieved depth value will be the same wherever the depth image is sampled. This does not hold for the edges of the surface. For the edges, the incorrectness of the sample location causes a slight depth distortion. The incorrectness depends on the depth of the single sample on the ray. Because the depth of the sample on the ray is chosen to be high, edges with a high depth will be correct. Near edges will have a slight depth distortion, as demonstrated in Section 7. Very near objects should already be avoided because of their larger area with missing information, so the effect of the depth distortion on edges is small.

The resulting algorithm can also be interpreted in a different way: The depth image is sampled at a constant offset. The resulting depth value is used as the depth for reprojection, as described in Section 2.2.

The source code of this implementation is available at http://www.marries.nl/projects/stereoscopic-3d/.
Algorithm 1 Gather-based stereo reprojection shader for one eye (pseudo-code).

---
Vertex shader:
Inverse (off-axis) project the current screen coordinate
to obtain the direction of the ray.
Normalize the direction to have a unit z-coordinate.
Calculate the position on the ray for the single sample.
Project the position on the ray to find the depth image
sample location.

---
Pixel shader:
Sample the depth image to obtain the final ray depth.
Calculate the position on the ray at the final ray depth.
Project the position on the ray.
Sample the color image at the projected position.

5.3 Generalized algorithm
The generalized algorithm uses ray marching, in contrast to the algorithm presented in Section 5.2. This solves the problem of depth distortion at near edges at the cost of a longer execution time. The following steps are taken in the algorithm:

- The offsets for the samples are pre-calculated.
- The z-normalized direction of the ray is calculated in the vertex shader.
- The iterative ray march is done in the pixel shader:
  - Retrieve depth $d_{\text{image}}$ from the depth image at the sample offset of
    the current iteration.
  - Compare $d_{\text{image}}$ to the ray depth $d_{\text{ray}}$ of the current iteration.
  - End the ray march if $d_{\text{ray}} > d_{\text{image}}$. (The intersection depth is
    approximated by using $d_{\text{image}}$ as the intersection depth.)
- Find the intersection position on the ray with depth $d_{\text{image}}$.
- Project the intersection position to the center camera.
- Sample the color image at the projected position.

6 Benchmarks
Our benchmarking implementation uses OpenGL 3.1 and NVIDIA Cg 3.0.0.15. On AMD hardware we used GPUPerfAPI, which utilizes internal GPU timers.
On NVIDIA hardware we used the CPU timer, which was synchronized with the GPU using the glFinish call. The glFinish call introduces overhead which is also captured in the measurements.

The benchmark was performed at a resolution of 1280 × 720. It uses the algorithm as described in Section 5.2 and includes two reprojection passes, one for each eye. The results were averaged over 60 subsequent frames. Three different setups were used for the benchmark:

1. ATI Radeon HD 5850 (driver version 8.831.2), 4GB RAM, Intel Core 2 Quad Q6600.
2. ATI Radeon HD 5670 (driver version 8.74), 4GB RAM, AMD Phenom II X6 1035T.
3. NVIDIA GeForce 9600 GT (driver version 8.17.12), 2GB RAM, Intel Core 2 Duo E8400.

The different CPU’s have a minimal effect on the measurements because the benchmark is fully GPU bound.

The results of the benchmark are shown in Table 1.

### Table 1: Benchmark results.

<table>
<thead>
<tr>
<th></th>
<th>Reprojection (left+right)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HD 5850</td>
<td>0.228 ms</td>
</tr>
<tr>
<td>HD 5670</td>
<td>0.453 ms</td>
</tr>
<tr>
<td>9600 GT</td>
<td>0.780 ms</td>
</tr>
</tbody>
</table>

7 Quality Results

The quality is affected by three main factors:

**The quality of the reconstructed depth** affects the accuracy of the reprojection at edges. This differs between the Algorithm presented in Section 5.2 and the Generalized Algorithm presented in Section 5.3.

**The area with missing information** is an inherent effect of reprojection which introduces an area with unknown information. This cannot be resolved without additional information. The entire area with missing information will be different from the correct area.

**The image resampling** causes small differences in high frequency areas. The type of differences can be changed by sampling the image with bilinear filtering as explained in Section 4.3.
Figure 10: Image comparison of the left eye image.

Figure 11: Image comparison of the left eye image in a situation with very near objects.
### 7.1 Color comparison

Figures 10 and 11 compare reference images with reprojected images. Figure 10 demonstrates the reprojection quality with positive parallax, while Figure 11 shows the reprojection quality with a negative parallax. The area with missing information is large in Figure 11 because the pole is very close to the camera; it comes out of the screen. The area with missing information is at the left side of the pole. The difference at the right side of the pole in the Reprojection Algorithm is caused by the depth distortion due to the incorrectness of the single sample position.

A pixel is marked red in Figures 10 and 11 if the value changed more than 0.05 (averaged over the color channels), where each channel is in the range $[0\ldots1]$. The stereoscopic parameters used to generate the images cannot be described easily because there is no independent metric for stereoscopic parameters. However, the used parameters can be judged from the anaglyph images in Figure 13 and 14. The stereoscopic parameters are chosen to be comparable to existing stereoscopic games.

The mean squared error between the full images is shown in Table 2. Bilinear filtering has a larger impact on improving the similarity than a better depth reconstruction in both views.

The main difference between the views in Figure 10 and 11 is the total size of the areas with missing information. This results in a higher MSE for the view in Figure 11.

### 7.2 Depth comparison

Figure 12 shows the quality of the reconstructed depth. A pixel is marked red in Figure 12 if the intensity changed more than 0.01, where the intensity is in the range $[0\ldots1]$. The difference between the Reprojection Algorithm and the Generalized Reprojection Algorithm is large. The Generalized Algorithm is a nearly perfect reprojection of the depth, except for the area with missing information. The Reprojection Algorithm has more depth distortions at the edges.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>View in Figure 10</th>
<th>View in Figure 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reprojection Algorithm</td>
<td>0.000937</td>
<td>0.001087</td>
</tr>
<tr>
<td>Generalized Reprojection Algorithm</td>
<td>0.000778</td>
<td>0.000961</td>
</tr>
<tr>
<td>Reprojection Algorithm (Bilinear filtered)</td>
<td>0.000596</td>
<td>0.000735</td>
</tr>
</tbody>
</table>

Table 2: The MSE between the reference image and the reprojected images.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Depth in Figure 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reprojection Algorithm</td>
<td>0.000126</td>
</tr>
<tr>
<td>Generalized Reprojection Algorithm</td>
<td>0.000068</td>
</tr>
</tbody>
</table>

Table 3: The MSE between the reference depth and the reprojected depth.
Figure 12: Comparison of the reconstructed depth for the left eye in a situation with very near objects.

Table 3 shows the mean squared error between the depth images. The relative difference between the results is much larger than the corresponding difference in the color image comparison. Apparently, the absolute correctness of the depth is only a small factor in the correctness of the color image. In applications where the execution time is important, the algorithm presented in Section 5.2 should be favored over the generalized algorithm of Section 5.3.

8 Conclusion

We have presented a very fast reprojection technique which generates high quality stereoscopic images. The performance difference is enormous compared to the alternative of rendering everything twice. The difference will grow in the future because the performance of reprojection only depends on the resolution, whereas the performance of rendering twice depends on the complexity of the entire rendering stage.

The disadvantage of reprojection is the small reduction in image quality, but this is only noticeable when objects are very close to the camera.

Acknowledgements

We would like to thank Wolfgang Hürst, Arno Kamphuis and Cor Jansen for making this research project possible. Furthermore, we want to thank Crytek for publishing the Sponza Atrium scene.
References

http://www.crytek.com/sites/default/files/Part 2 - Bringing Stereo to Consoles.ppt


http://repository.tudelft.nl/view/ir/uuid%3Ae551171e-ac6d-470f-a29a-193c7e9f2ace/
A Anaglyph images

Figure 13: The scene used in Figure 10 rendered using the Reprojection Algorithm from Section 5.2.

Figure 14: The scene used in Figure 11 rendered using the Reprojection Algorithm from Section 5.2.