Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

Terrain Rendering using Spherical Clipmaps

Malte Clasen and Hans-Christian Hege

Zuse Institute Berlin, Germany

Abstract

We describe a terrain rendering algorithm for spherical terrains based on clipmaps. It leverages the high geometry throughput of current GPU to render large static triangle sets. The vertices are displaced by a height map texture. Our main contribution is mapping of texture coordinates to calculate the height map sample position based on the static vertex offset and the variable view position.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

1. Introduction

Terrain rendering has a broad range of applications from science, e.g. cartography and landscape planning, to entertainment, e.g. outdoor games and movies. We focus on the serious applications that usually don't allow artistic tricks to hide technological deficiencies. The target is quite simple to state: We want to visualize spherical terrains (whole planets) on many scales (from space flight to sunday afternoon walk) on commodity hardware. This imposes two major challenges: The size of the data exceeds the capabilities of current PCs by far and numerical errors of 32 bit floating point numbers, the maximum accuracy of current GPU, become relevant.

2. Existing technology

Previous publications and applications can be divided into two parts: Those with planar terrain and those with sperical terrains. Both converge to the same solution with increasing scale, and there are many cases where a planar terrain is absolutely sufficient. But in the real world, you just cannot see from Lisbon to New York.

2.1. Planar terrain

Many popular terrain rendering algorithms deal with planar terrain. Losasso and Hoppe categorize them as follows [LH04]:

- Irregular meshes (a.k.a. triangulated irregular networks)
- Bin-tree hierarchies (a.k.a. longest-edge bisection, restricted quadtree, hierarchies of right triangles)
- Bin-tree regions (coarser than Bin-tree hierarchies)
- Tiled blocks (square patches that are tessellated at different resolutions)

The error for a given number of triangles increases with each category. Irregular meshes result in the best possible approximation but requires a large computational overhead. In practice, tiled block algorithms can take advantage of the huge geometry bandwidth of current GPU most effectively and overcompensate their deficiencies in accuracy. Losasso introduces the Geometry Clipmaps algorithm in [LH04] which is especially designed for this bandwidth. Asirvatham and Hoppe further improve it in [AH05] to handle most of the computations on the GPU.

2.2. Spherical terrain

Although the same categorization is valid for spherical terrain, most research seems to focus on planar terrain. O'Neil [O'N01] and Hill [Hil02] tried to extend the ROAM algorithm [DWS*97] (bin-tree hierarchy) to handle spherical surfaces, but Hill dropped this approches in favor of a tiled block solution in the same publication. Cignoni et.al. introduce a bin-tree region type algorithm, they extend the BDAM algorithm [CGG*03a] to planets (P-BDAM) [CGG*03b]. All solutions have in common that they partition the planet into square regions, using a cube as base geometry.

The popular terrain viewers Google Earth (`http://`

earth.google.com/) and NASA World Wind (http://worldwind.arc.nasa.gov/) apparently use tiled block approaches, but these solutions are not published.

3. Spherical clipmaps

We chose to extend the GPU-based geometry clipmaps by Asirvatham to spherical terrains because of the following reasons:

- The rendering speed depends on the screen resolution, not on the size of the digital elevation model (DEM) and the corresponding surface color texture. This basic feature of each LOD algorithm is handled exceptionally well by the underlying clipmap. Image resampling is a thoroughly researched domain and this knowledge can be applied directly in the construction of the clip map.
- Different levels of detail can be blended smoothly even when they are more than one level apart.
- Rendering can be limited on-the-fly to the coarsest n levels without overhead in case streaming data is late or the framerate does not meet the requirements.
- The implementation is simple because the geometry is static and the only image operation is copying regions between buffers.
- The technique is quite fast and the current bottleneck, vertex texture look-ups, is expected to disappear with unified shaders.

The following changes to the original algorithm enable spherical terrains:

3.1. Clipmaps

The original Clipmap by Tanner [TMJ98] is a texture representation that can be used to display textures of virtually unlimited size with maximum detail around a variable focus point. It resembles a mipmap pyramid where each level is clipped to a fixed number of samples around the focus point (fig. 1). When a level is sampled, it is first tested whether the sampling point lies in the clipped region. If not, the next higher level is searched, which covers an area four times as large. This results in a memory requirement of $O(\log n)$ for a base texture of size n.

Losasso and Hoppe used this representation for height maps. This effectively enables the usage of arbitrary height map sizes independent of run-time memory requirements and provides an inherent level of detail representation that reduces rendering time similar to memory. Each ring is rendered using the same number of vertices just as each ring contains the same number of image samples.

Since the main feature of Geometry Clipmaps is the static geometry relative to the viewer (plus some minor translation), this support geometry had to be changed to accommodate our parametrisation: Any rectangular grid aligned to the underlying parametrisation changes its shape with the distance to the poles of the planet. The problems becomes inevitably visible when the viewer is close to the pole: The support geometry becomes infinitely thin towards the pole and stops there as spherical coordinates do not wrap around in θ direction.

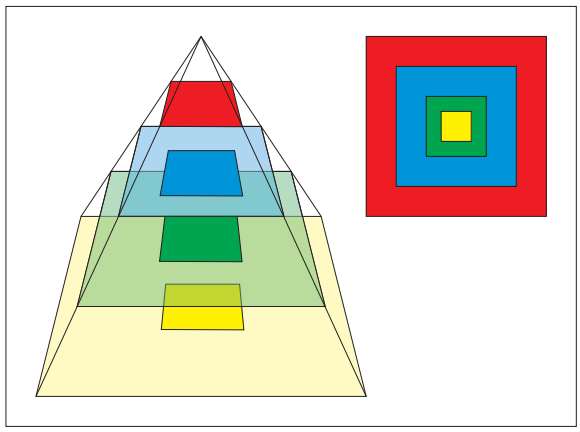


Figure 1: *The clipmap contains a fixed-size segment of each mipmap level around an arbitrary focus point.*

In the following we replace the underlying geometry with one that maps better to the sphere. No matter how far away the viewer is relative to the planet, he cannot see more of it than one hemisphere. This led to the idea of using concentric rings instead of rectangles. The resulting spherical Geometry Clipmap is displayed in fig. 2.

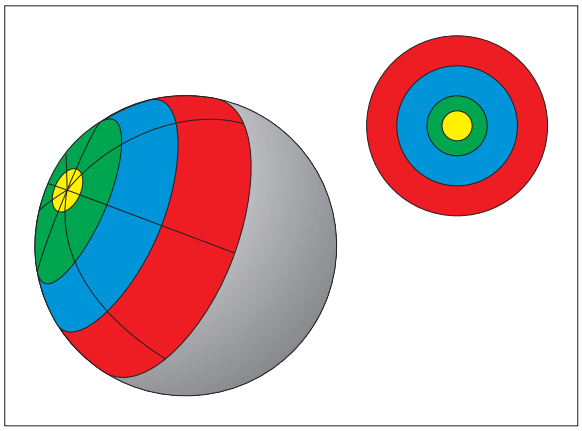


Figure 2: *We use circular instead of rectangular rings to cover the hemisphere.*

3.2. Map parametrisation

The trivial parametrisation of the plane, (x,y), cannot be transfered directly to the sphere. However, an equally simple parametrisation exists: Spherical coordinates, denoted by

$(\phi,\theta) \in [0,2\pi) \times [0,\pi)$ (fig. 3). Given a coordinate system with the axes (x,y,z), a point p on the unit sphere can be parametrized by its angle *theta* to the z-axis, and the angle *phi* from p projected to the x,y-plane to the x axis. $(0,0,1)$ and $(0,0,-1)$ can be denoted as north and south pole respectively. All points with the same *phi* belong to a meridian, the 0-meridian intersects the positive x-axis.

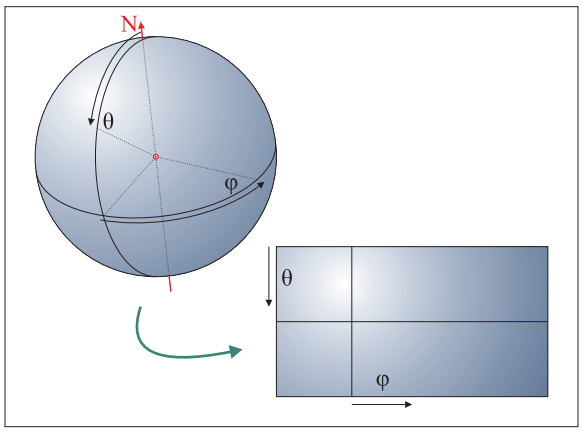


Figure 3: *The sphere can be parametrized by* (ϕ,θ) *which map directly to a planar rectangle.*

3.3. Map transform

Since we want to focus the clipmap around the viewer, we have two different spaces: The world space (x,y,z) that provides an absolute orientation of the spherical terrain and the view space $(\tilde{x},\tilde{y},\tilde{z})$ that locates the viewer at the north pole. The introduction of the view space enables a static geometry (vertices plus connectivity) that has to be calculated and transferred to the GPU only once. The hemisphere around the viewer is parametrized by $(\tilde{\phi},\tilde{\theta})$ whereas the terrain is parametrized by (ϕ,θ). The mapping between both spaces (fig. 4) depends on the position of the viewer (in world space), $(\phi_\mathbf{v},\theta_\mathbf{v})$.

We can assume without loss of generality that the viewer v is located exactly above the 0-meridian at $(0,\theta_\mathbf{v})$ since any deviation in $\phi_\mathbf{v}$ translates directly to a simple ϕ-offset in the height map. Thus we need a mapping

$$f(\theta_\mathbf{v},\tilde{\phi},\tilde{\theta}) \rightarrow (\phi,\theta). \tag{1}$$

This mapping is a rotation around the y-axis as we chose the 0-meridian to intersect the positive x-axis (fig. 5).

A point $\tilde{p}$ on the hemisphere with the local spherical coordinates $(\tilde{\phi},\tilde{\theta})$ has the coordinates

$$\tilde{p} = \begin{pmatrix} cos\tilde{\phi} \cdot sin\tilde{\theta} \\ sin\tilde{\phi} \cdot sin\tilde{\theta} \\ cos\tilde{\theta} \end{pmatrix} \tag{2}$$

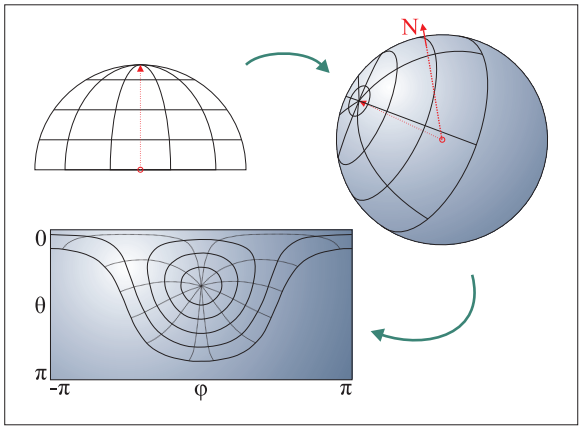


Figure 4: *Points on the view hemisphere are transformed into world space to sample the rectangular height map.*

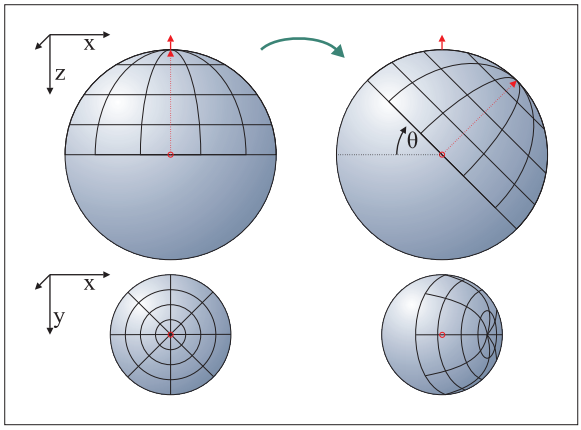


Figure 5: *For* $\phi_\mathbf{v} = 0$*, the hemisphere is rotated only around the y-axis.*

in view space. The rotation affects only the x and z coordinates, resulting in:

$$p = \begin{pmatrix} cos\theta_\mathbf{v} \cdot \tilde{p}_x - sin\theta_\mathbf{v} \cdot \tilde{p}_z \\ p_y \\ -sin\theta_\mathbf{v} \cdot \tilde{p}_x + cos\theta_\mathbf{v} \cdot \tilde{p}_z \end{pmatrix} \tag{3}$$

This point is converted back into spherical coordinates by:

$$\begin{pmatrix} \phi \\ \theta \end{pmatrix} = \begin{pmatrix} tan^{-1}\frac{p_y}{p_x} \\ cos^{-1}(p_z) - \theta_\mathbf{v} \end{pmatrix} \tag{4}$$

Note that we subtract $\theta_\mathbf{v}$ from θ to set the origin of the transformed coordinate system to the position of the viewer. This offset and the previously fixed $\phi_\mathbf{v} = 0$ define the focus point of the clip map. tan^{-1} has to take into account the quadrant in which (p_y,p_x) lies, similar to `atan2()` in C.

For a vertex on the hemisphere, we precalculate $\tilde{p}$ on the CPU and pass it as vertex attribute to the vertex shader where

the next two steps are performed. The per-frame-constants $cos\theta_{\mathbf{v}}$ and $sin\theta_{\mathbf{v}}$ can also be calculated on the CPU and passed as uniforms to the shader.

3.4. Discretization

The original Geometry Clipmaps use a rectangular support geometry that is aligned to the grid of the underlying raster data. Since rectangular grids are the native representation of textures on current GPU and also quite common in cartography and artistic terrain generation, we continue to use it although it does not allow a direct correspondence of vertices to height samples. We use spherical coordinates to transform map the height texture (parametrized by (s,t)) to the spherical surface: $(s,t) = (\phi,\theta)$.

The hemisphere $(\tilde{\phi},\tilde{\theta})$ is discretized into quads. $\tilde{\phi}$ is simply divided into n fixed steps. The discretization of $\tilde{\theta}$ depends on the distance to the viewer: Low levels of detail (far away) require less steps per distance than higher levels. The first level of discretization divides the hemisphere into the concentric rings that shrink exponentially: Level i covers $\tilde{\theta} \in (2^{-i}\pi, 2^{-i-1}\pi]$. This sequence is terminated by a fill level that covers $\tilde{\theta} \in (2^{-i-1}\pi, 0]$. Each level is subdivided into m rings by $\tilde{\theta}_{i,j} = \theta_{i,0} * 2^{-j/m}$. Each discrete element of the hemisphere is then partitioned into two triangles. The resulting geometry ensures that the triangles have about the same size in screen space (fig. 6).

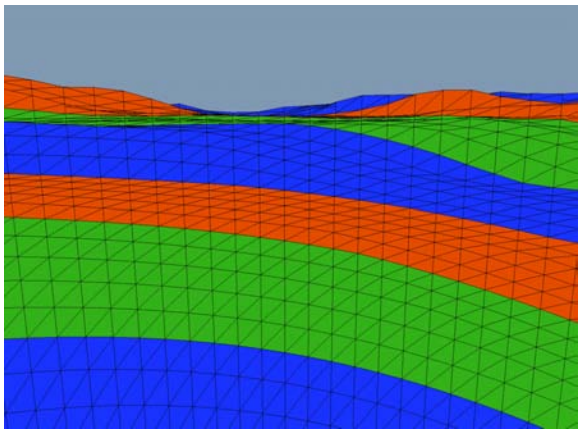


Figure 6: *Triangles twice as far away are rendered twice as small, so our expontially growing triangles have about the same size in screen space.*

The disadvantage of this solution compared to the original GPU-based Geometry Clipmaps is the 1:1 correspondence of vertices and height samples had to be dropped. One advantage is that no special case handling is required to circumvent the T-intersections at level boundaries: The constant discretization of $\tilde{\phi}$ implies the gapless geometry transition.

4. Algorithmic details

There are few more algorithmic differences to Asirvatham's Geometry Clipmaps that are caused by the new mapping:

4.1. Texture sizes in map space

The size of a support geometry level in map space is now dependent on the position of the viewer: A circular ring with a diameter of $1m$ covers a ϕ range of about $\frac{2\pi}{40{,}000}$ if the viewer is located at the earth equator. The same ring covers the whole 2π if the viewer is standing less than $1m$ away from the north pole. Therefore the map space range of the clipmap texture has to be chosen according to the current $\phi_{\mathbf{v}}$. There is no direct dependence on $\theta_{\mathbf{v}}$ apart from the fact that the texture can be clipped at $\theta < 0$ and $\theta > \pi$ since map sampling does not cross the poles.

This anisotropic range is shown in fig. 7. The level covers a map range of $\phi = 2\pi$ if the level diameter $\tilde{\theta}$ is less than the distance to the nearest pole:$(\tilde{\theta} > \theta_{\mathbf{v}}) \vee (\tilde{\theta} > \pi - \theta_{\mathbf{v}})$.

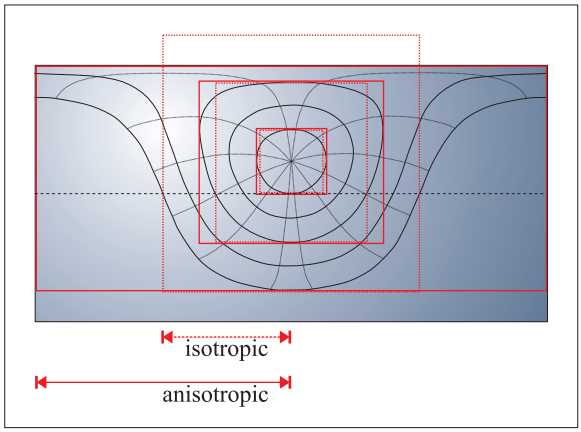


Figure 7: *The world space size of the clipmap regions have to be chosen according to $\theta_{\mathbf{v}}$ to handle the anisotropy.*

One of the advantages of the wrap-around clipmap updates is that small movements of the viewer cause small texture updates. If we use the texture completely for any given ϕ-range, a small movement of $\theta_{\mathbf{v}}$ would require an update of the whole texture. Therefore we change the texture range in power of two steps and use only a subset of the texture. For instance ϕ-ranges of 0.3π and 0.4π both result in a texture that covers 0.5π. This results in an ammortized complexity that matches the original Geometry Clipmaps.

4.2. Aliasing

The missing direct correspondence of height map samples to vertices introduces a possible source of aliasing: The base signal (height map) is resampled at a different rate by the support geometry. The triangles equal a linear interpolation

which is a less than perfect reconstruction filter. Resampling at a frequency that is at least as high as the sampling frequency of the source signal ensures that the aliasing is minimized and no detail is lost , but the lower the source signal rate, the lower the visual details of the terrain. A good choice is a resampling rate that roughly equals the source sampling rate: If you discetrize $\tilde{\phi}$ into n steps, then a texture width of $\frac{n}{4}$ is sufficient since this texture has about n boundary texels. Any texture size between this upper bound and $\frac{n}{8}$ should be fine, assuming that $\tilde{\theta}$ is discretized at a similar resolution.

4.3. Clipmap filtering

The clipmap pyramid is based on successively downscaled images. This scaling is performed in map space, but resampling by the support geometry vertices is performed in 3D world space. This introduces another possible source of aliasing since the source density in ϕ direction at the poles is far higher than at the equator. Having the same number of ϕ samples is merely an artefact of the chosen parametrisation, so the signal bandwidth has to be limited artificially. This can be done by using a special filter kernel: Common image resampling algorithms use circular kernels. They match the circular shape of the support geometry, so the same strategy can be applied. The filter kernel should be defined in 3D world space and transformed to map space as described in 3.3. This way the bandwidth is limited so that the resampling by the support geometry works as expected.

4.4. Texture coordinates beyond poles

The arcus tangent in the calculation of p'' in 3.3 works based on the assumption that the map wraps around in ϕ direction: The texture coordinates decrease towards $\phi = -\pi$ and increase towards $\phi = +\pi$. They meet at the π-meridian exactly beyond the pole. There's no problem mathematically, but the discretization causes an artefact at that point. The texture coordinates are interpolated across the triangles, so the last triangle in one of the two directions interpolates from ε to 1 instead to 0 (fig. 8, left).

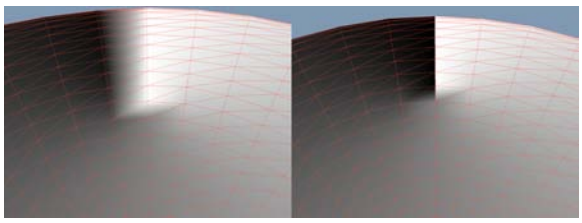


Figure 8: *Texture coordinate interpolation results in artefacts beyond the poles, so one line of vertices has to be duplicated.*

This can be fixed by duplicating the vertices on that meridian: Since the support geometry is always oriented with $\tilde{\phi} = 0$ in north direction, only one line of vertices requies special handling. One vertex of each pair gets a special attribute that is used in the vertex shader to correct the texture coordinate. We determine whether the pair lies beyond a pole (compare to θ_v) and subtract 1 from the calculated texture ϕ coordinate, so the interpolation across the triangles works as expected (fig. 8, right).

4.5. Level visibility

Not all circular rings of the hemisphere are visible from each position. The lower bound (low detail, far away) is determined by the earth curvature, the upper bound (high detail, near) by the height above the local surface. The lower bound can be estimated as shown in fig. 9: The height h of the viewer above the spherical planet surface (radius r) determines the tangent cone to the planet. The terrain beyond this $\tilde{\theta}_{max}$ is hidden by the earth curvature (note that the minimum level of detail corresponds to the maximum $\tilde{\theta}$):

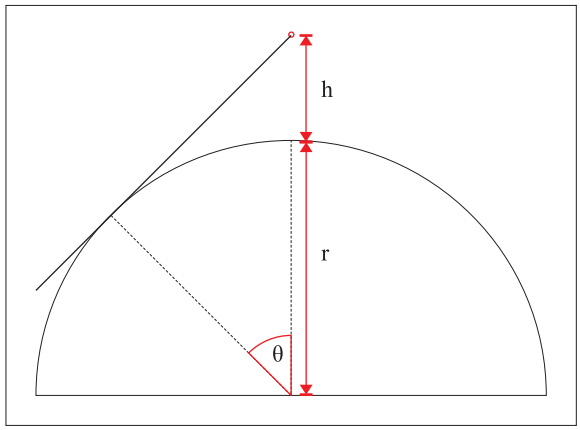


Figure 9: *Visibility of the lower levels of detail depends primarily on earth curvature and the distance to the surface.*

$$(r+h)\cdot cos\tilde{\theta}_{max} = r \quad (5)$$

$$\Leftrightarrow \tilde{\theta}_{max} = cos^{-1}\frac{r}{r+h} \quad (6)$$

This calculation does not take the slope of the terrain into account (e.g. high mountains might be clipped early), so you might want to add a safety factor to this approximation.

The upper bound is calculated based on the requirement that triangles should cover at least one pixel in screen space. The size s of one screen pixel on the surface of the planet depends on the height h of the viewer, the field of view angle fov and the number of pixels $\#p$ per scanline (fig. 10):

$$s \approx h \cdot tan\frac{fov}{\#p} \quad (7)$$

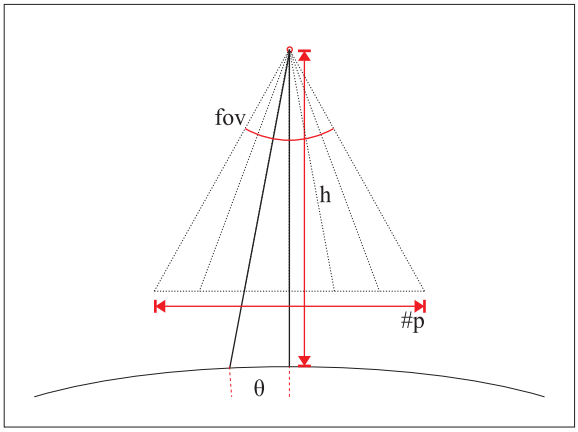


Figure 10: *Visibility of the higher levels of detail depends on the screen resolution and the distance to the surface.*

The upper bound $\tilde{\theta}_{min}$ follows directly:

$$\tilde{\theta}_{min} = \frac{s}{2\pi r} \cdot 2\pi = \frac{s}{r} \tag{8}$$

5. Implementation

The following aspects deal with the implementation on current consumer GPUs. Using vertex texture look-ups currently limits the technique to NVIDIA NV40 and G70 class GPU (Geforce 6600, 6800, 7800) since ATI does not support this feature up to the R520 line (Radeon X1800). A possible work-around is the render to vertex buffer support that allows using the pixel shader to calculate the actual vertex positions in a pre-pass. We focussed on the NV40 and found the following issues:

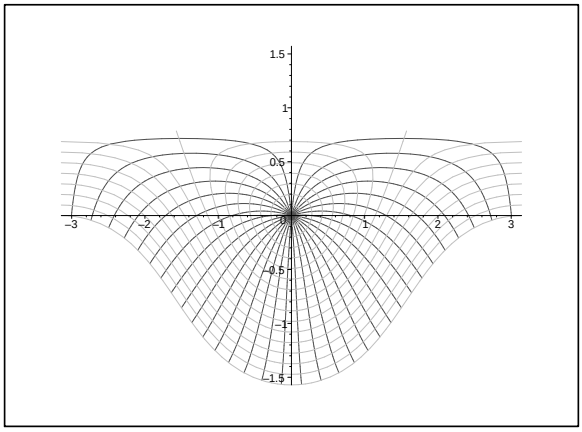
5.1. Trigonometric function replacement

Calling trigonometric functions in the vertex shader is a possible bottleneck. Our map transform algorithm relies on tan^{-1} and cos^{-1} that cannot be precalculated efficiently. But there's another way out: The distortion of the circular ring in map space is quite low for higher levels of detail (small $\tilde{\theta}$). Figure 11 illustrates this for $\tilde{\theta} < \frac{\pi}{128}$ and $\theta_v = \frac{3}{8}\pi$. These inner rings can be transformed using a simple approximation for p''_ϕ:

$$\phi = tan^{-1}\frac{p_y}{p_x} \tag{9}$$

$$= tan^{-1}\frac{\tilde{p}_y}{cos\theta_v \cdot \tilde{p}_x + sin\theta_v \cdot \tilde{p}_z} \tag{10}$$

$$\approx \tilde{p}_y \cdot \left(1 + \frac{\frac{1}{\pi}}{\frac{1}{4} - (\frac{\theta_v}{\pi} - \frac{1}{2})^2} - \frac{(\frac{\theta_v}{\pi} - \frac{1}{2})^2}{6} - \frac{4}{\pi}\right) \tag{11}$$



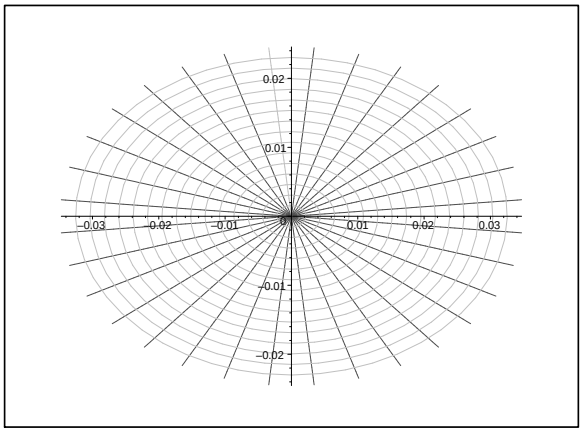


Figure 11: *$\tilde{\phi}$- and $\tilde{\theta}$-iso-lines in world space (ϕ, θ): Whereas the overall distortion of the mapped hemisphere is quite large, the area around the viewer is only stretched in ϕ-direction.*

The distortion term depends only on θ_v and can thus be precomputed on the CPU. This empirically derived approximation is tuned to the following setting: $\tilde{\theta}$ should be small, for instance $< \frac{\pi}{1024}$, and θ_v should be not too near to the poles, e.g. $\pi\frac{1}{48} < \theta_v < \pi\frac{47}{48}$. These limits result in an relative approximation error $1 - \frac{approximated}{exact}$ (fig. 12) less than 0.001 for $\tilde{\phi} = \frac{\pi}{2}$. We consider this value acceptable for interactive rendering.

Apart from the slow computation, tan^{-1} has another drawback: The accuracy of `atan(y,x)` on the NV40 is quite limited for small `x`, so higher levels of detail show significant errors in the ϕ texture coordinate (fig. 13). The

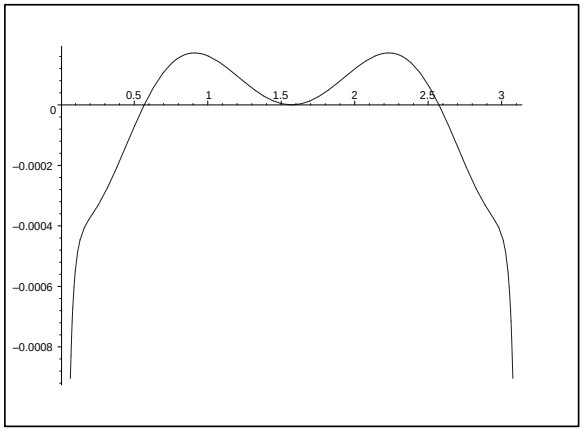


Figure 12: *The relative approximation error stays below* 0.001 *for* $\pi\frac{1}{48} < \theta_{\mathbf{v}} < \pi\frac{47}{48}$

simple solution is to use the approximation formula at least starting at the levels that exhibit the incorrect behaviour.

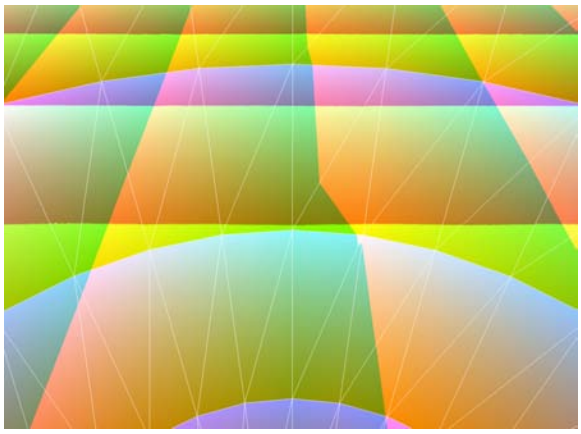


Figure 13: *The inaccuracy of the* tan^{-1}*-implementation causes distortion in the texture coordinate calculation: The central* ϕ *line should be straight, not jagged.*

As motivated above, the inner rings resemble a stretched circle. Therefore the θ-direction requires no further calculations and can be approximated as follows:

$$\theta = cos^{-1}(p_z) - \theta_{\mathbf{v}} \quad (12)$$

$$= cos^{-1}(-sin\theta_{\mathbf{v}} \cdot \tilde{p}_x + cos\theta_{\mathbf{v}} \cdot \tilde{p}_z) - \theta_{\mathbf{v}} \quad (13)$$

$$\approx \tilde{\theta} \quad (14)$$

This approximation is also usable under the previously mentioned conditions. If we take all possible view positions into account, $\theta_{\mathbf{v}}$ is relatively large compared $\tilde{\theta}$ in the higher levels of detail in most cases (except close to the north pole). Therefore the numerical error in the calculation of θ introduced by the difference $cos^{-1}(p_z) - \theta_{\mathbf{v}}$ dominates the error of this approximation, so this very simple approximation suffices. Nevertheless the result is visually acceptable.

Note that you should blend from approximated to exact calculation to avoid gaps in the terrain (fig. 14).

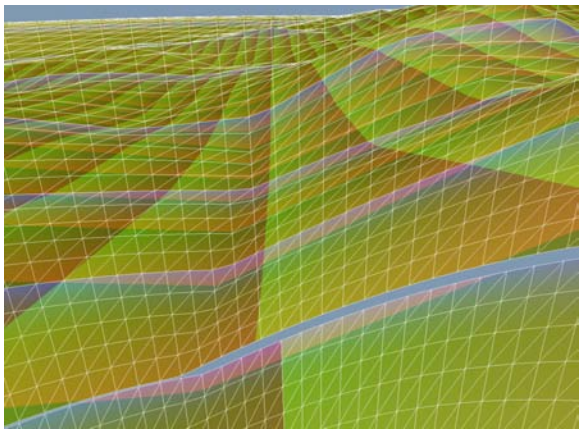


Figure 14: *Missing blending between exact calculation and approximation can lead to gaps.*

5.2. Speed

The main bottleneck is the vertex texture look-up: Since we had to drop the 1:1 correspondence, we have to use texture filtering to avoid the strong artefacts of nearest neighbor sampling. The NV40 is not capable of filtering vertex textures, but bilinear filtering can be emulated using 4 samples. The blending region between two levels of detail requires trilinear filtering (8 samples). This overhead hides any other possible bottlenecks, even the trigonometric functions do not affect the framerate in this case. It would be a major limitation of the whole technique, but we expect the vertex texture lookups to improve as soon as unified shader architectures become widespread: Common pixel shader units are especially designed to deal with the latency of texture lookups. Implementing this into vertex units in separated architectures would increase the chip complexity disproportionately for a feature that is rarely used in current games. With a unified shader architecture, vertex shaders could use the same technology almost for free, so we believe that this bottleneck will disappear in the next one or two years.

We benchmarked the algorithm on a Pentium 4 (2.4 GHz) system with NVidia NV40 GPU (325 MHz, Geforce 6800). The screen resolution of 1280×960 was no bottleneck since our implementation is vertex shader limited. The test data set consisted of a height map with 43200×21600 pixels (338 MB JPEG 2000 compressed) and a color map with

86400×43200 pixels (203 MB ECW compressed). The clipmap texture sizes for height map and color map are 128^2 and 512^2 respectively. $\tilde{\phi}$ and $\tilde{\theta}$ are discretized into 512 steps (20 $\tilde{\theta}$-steps per level). The approximative transform was used for levels $>= 10$. In this configuration, the following views were used: A overview over Lake Garda (Italy) from the south east with the camera standing on the ground, see fig. 15. Levels 6 to 21 were rendered (about $5.2 \cdot 10^6$ triangles per frame) at 25 frames per second.

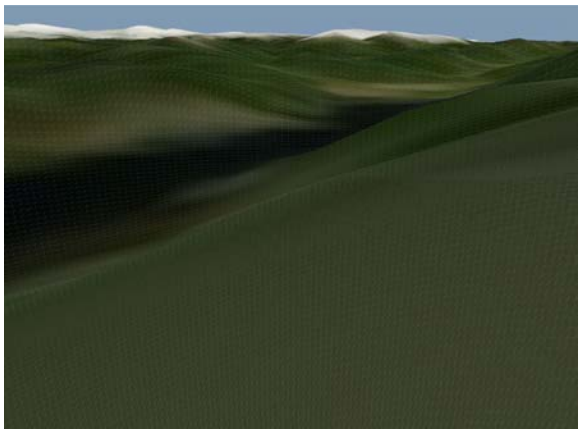


Figure 15: *Lake Garda, ground view*

The second view shows the same area from an aircraft perspective (fig. 16). Levels 3 to 9 were rendered ($2.3 \cdot 10^6$ triangles) at 40fps.

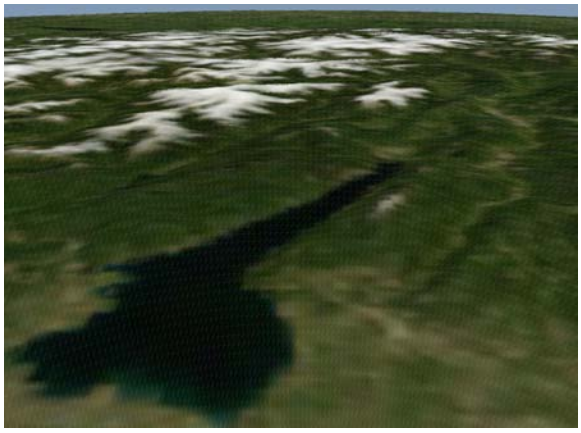


Figure 16: *Lake Garda, aircraft view*

Increasing the altitude again resulted in the third test case, the space view (fig. 17). Levels 1 to 4 were rendered ($1.3 \cdot 10^6$ triangles) at 65fps.

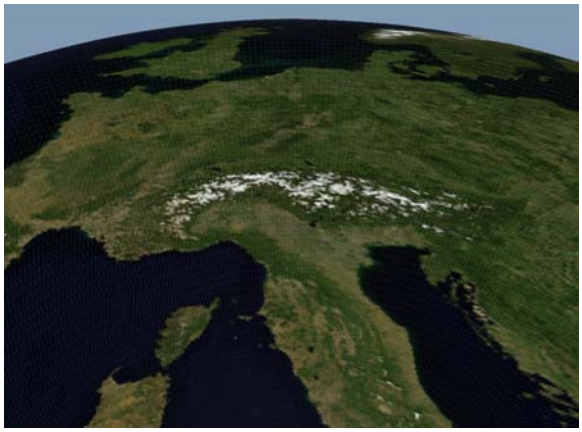


Figure 17: *Lake Garda, space view*

6. Conclusions

We presented an extension to the GPU-based Geometry Clipmaps by Asirvatham et.al. that handles spherical terrains. It performs well for a large range of view conditions from space (fig. 17) over aircraft heights (fig. 16) to a stroller's perspective (fig. 15). The implementation is simple and the special cases (texture coordinates beyond poles, arcus tangent accuracy) can be handled in a few lines of code. Additional textures such as color map and normal map can be handled using the same implementation without additional effort.

6.1. Future work

Our implementation currently lacks any view frustum culling. Adding this can result in a speed up of factor 8 for a field of view of $\frac{\pi}{2}$ since the main bottleneck, the vertex texture lookups, scales linearly.

The addition of render to vertex buffer support is another target. We don't expect algorithmic changes but the bottlenecks might shift to previously unconsidered aspects. A major advantage of this solution would be that the geometry clipmap had to be evaluated only once per change of the position of the viewer, not once per frame. This should improve the performance for multipass renderings as required for shadow mapping with one or more light sources.

7. Acknowledgement

We would like to thank the Bundesministerium für Bildung und Forschung (BMBF, http://www.bmbf.de/) for supporting the SILVISIO project (FKZ 0330560B) for which this terrain rendering algorithm has been developed. We also appreciate the publication of the Blue Marble texture set and the Shuttle Radar Topography Mission data by NASA (http://earthobservatory.nasa.gov/).

References

[AH05] ASIRVATHAM A., HOPPE H.: *GPU Gems 2*. Addison-Wesley, 2005, ch. Terrain Rendering Using GPU-Based Geometry Clipmaps, pp. 27–46. http://research.microsoft.com/~hoppe/.

[CGG*03a] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Bdam – batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum 22 (3)* (2003). http://vr.c-s.fr/vplanet/Publications/Papers/eg2003-bdam.pdf.

[CGG*03b] CIGNONI P., GANOVELLI F., GOBBETTI E., MARTON F., PONCHIO F., SCOPIGNO R.: Planet-sized batched dynamic adaptive meshes (p-bdam). In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), IEEE Computer Society, p. 20.

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D. E., MILLER M. C., ALDRICH C., MINEEV-WEINSTEIN M. B.: Roaming terrain: real-time optimally adapting meshes. In *VIS '97: Proceedings of the 8th conference on Visualization '97* (Los Alamitos, CA, USA, 1997), IEEE Computer Society Press, pp. 81–88. http://www.llnl.gov/graphics/ROAM/roam.pdf.

[Hil02] HILL D.: *An efficient, hardware-accelerated, level-of-detail rendering technique for large terrains*. Master's thesis, Graduate Department of Computer Science, University of Toronto, 20002. http://www.magma.ca/~dhlf/downloads/thesis.pdf.

[LH04] LOSASSO F., HOPPE H.: Geometry clipmaps: terrain rendering using nested regular grids. In *Siggraph 2004* (New York, NY, USA, 2004), vol. 23 (3), ACM Press, pp. 769–776. http://research.microsoft.com/~hoppe/.

[O'N01] O'NEIL S.: Rendering planetary bodies. *Gamasutra August 10, 2001* (2001). http://www.gamasutra.com/features/20010810/oneil_01.htm.

[TMJ98] TANNER C. C., MIGDAL C. J., JONES M. T.: The clipmap: a virtual mipmap. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), ACM Press, pp. 151–158. http://www.cs.virginia.edu/~gfx/Courses/2002/BigData/papers/Texturing/Clipmap.pdf.