StuCoSReC

Proceedings
of the 2018
5th Student
Computer
Science
Research
Conference

StuCoSReC

Proceedings of the 2018 5th Student Computer Science Research Conference

Edited by

Iztok Fister jr. and Andrej Brodnik

Reviewers and Programme Committee

Klemen Berkovič • University of Maribor, Slovenia
Lucija Brezočnik • University of Maribor, Slovenia
Patricio Bulić • University of Ljubljana, Slovenia
Zoran Bosnić • University of Ljubljana, Slovenia
Borko Bošković • University of Maribor, Slovenia
Janez Brest • University of Maribor, Slovenia
Andrej Brodnik • University of Primorska and University
of Ljubljana, Slovenia
Haruna Chiroma • Gombe State University, Nigeria
Mojca Ciglarič • University of Ljubljana, Slovenia
Jani Dugonik • University of Maribor, Slovenia
Iztok Fister • University of Maribor, Slovenia
Iztok Fister Jr. • University of Maribor, Slovenia
Matjaž Gams • Jozef Stefan Institute, Slovenia
Tao Gao • North China Electric Power University, China
Andres Iglesias • Universidad de Cantabria, Spain
Branko Kavšek • University of Primorska, Slovenia
Miklos Kresz • University of Szeged, Hungary
Niko Lukač • University of Maribor, Slovenia
Uroš Mlakar • University of Maribor, Slovenia
Marjan Mernik • University of Maribor, Slovenia
Eneko Osaba • University of Deusto, Spain
Vili Podgorelec • University of Maribor, Slovenia
Peter Rogelj • University of Primorska, Slovenia
Xin-She Yang • Middlesex University, United Kingdom
Damjan Vavpotič • University of Ljubljana, Slovenia
Grega Vrbančič • University of Maribor, Slovenia
Aleš Zamuda • University of Maribor, Slovenia
Nikolaj Zimic • University of Ljubljana, Slovenia
Borut Žalik • University of Maribor, Slovenia

Published by

University of Primorska Press
Titov trg 4, SI-6000 Koper

Editor-in-Chief

Jonatan Vinkler

Managing Editor

Alen Ježovnik

Koper, 2018

ISBN 978-961-7055-26-9 (pdf)

www.hippocampus.si/ISBN/978-961-7055-26-9.pdf

ISBN 978-961-7055-27-6 (html)

www.hippocampus.si/ISBN/978-961-7055-27-6/index.html

DOI: <https://doi.org/10.26493/978-961-7055-26-9>

© University of Primorska Press

Kataložni zapis o publikaciji (CIP) pripravili v Narodni in
univerzitetni knjižnici v Ljubljani

COBISS.SI-ID=296769024

ISBN 978-961-7055-26-9 (pdf)

ISBN 978-961-7055-27-6 (html)

Preface

It is no longer necessary to emphasize that computer science is omnipresent today. Terms as digitalization, Industry 4.0 etc. are frequently heard in the mass media and are becoming a part of our everyday life. The rapid advances in computer science require a large investment in research and development. In particular, it is important to educate young scientists, which is one of the objectives of the StuCoSReC conference.

In front of you is the proceedings of the fifth StuCoSReC conference. The StuCoSReC is intended for masters & PhD students to show their research achievements. Socializing is also important and the conference is an excellent opportunity for students to get to know each other. The conference connects students of computer science and goes beyond the invisible limits of faculties. The uniqueness of StuCoSReC conference is that it is organized each year by different higher education institution. The University of Ljubljana - Faculty of Computer and Information Science is proud to host the conference this year.

Twelve papers addressed this conference, covering several topics of the computer science. All the papers were reviewed by two international reviewers and accepted for the oral presentation.

ref. prof. dr. Nikolaj Zimic

Preface

II

Papers

<i>Regulacija plovca v zračnem tunelu (zračna levitacija) na osnovi mehke logike</i> • Primož Bencak, Dominik Sedonja, Andrej Picej, Alen Kolman, Matjaž Bogša	5–13
<i>Razpoznavanje literature v dokumentih in določanje njihovih sklicev v besedilu</i> • Matej Moravec, David Letonja, Jan Tovornik, Borko Boškovič	15–19
<i>Klasifikacija samomorilskih pisem</i> • Dejan Rupnik, Denis Ekart, Gregor Kovačevič, Dejan Orter, Alen Verk, Borko Boškovič	21–25
<i>Načrtovanje poprocesorja za pretvorbo NC/ISO G-kode v translacije za 3/5 osnega robota ACMA ABB XR701</i> • Gregor Germadnik, Timi Karner, Klemen Berkovič, Iztok Fister	27–33
<i>Using constrained exhaustive search vs. greedy heuristic search for classification rule learning</i> • Jamolbek Mattiev, Branko Kavšek	35–38
<i>Real-time visualization of 3D digital Earth</i> • Aljaž Jeromel	39–42
<i>Towards an enhanced inertial sensor-based step length estimation model</i> • Melanija Vežočnik, Matjaž Branko Jurič	43–46
<i>Breast Histopathology Image Clustering using Cuckoo Search Algorithm</i> • Krishna Gopal Dhal, Iztok Fister Jr., Arunita Das, Swarnajit Ray, Sanjoy Das	47–54
<i>Time series classification with Bag-Of-Words approach post-quantum cryptography</i> • Domen Kavran	55–59
<i>The problem of quantum computers in cryptography and post-quantum cryptography</i> • Dino Vlahek	61–64
<i>Named Entity Recognition and Classification using Artificial Neural Network</i> • Luka Bašek, Borko Boškovič	65–70
<i>Context-dependent chaining of heterogeneous data</i> • Rok Gomišček, Tomaž Curk	71

Real-time visualization of 3D digital Earth

Aljaž Jeromel

Faculty of Electrical Engineering and Computer
Science, University of Maribor
Koroška cesta 46, 2000 Maribor, Slovenia
aljaz.jeromel@student.um.si

ABSTRACT

A new method for real-time visualization of Earth in 3D is presented. The method dynamically calculates visualization data and works in several steps. Firstly, the vertex coordinates in 3D are calculated. Then, the UV coordinates for texturing are calculated from vertex' position. The vertices are connected into triangles and sent to the shader pipeline, where they are processed by vertex shader, geometry shader and fragment shader. The vertex shader transforms the vertices from the world space to screen space. Geometry shader corrects possible interpolation errors by inserting extra vertices where needed, while the fragment shader assigns the colour to resulting fragments. The main advantage of the proposed method over the state of the art methods is its simplicity. Experimental results have shown that the proposed method produces a high FPS (frames per second), which makes it suitable for real-time visualization.

Keywords

Computer Graphics, 3D Rendering, Object Geometry

1. INTRODUCTION

With the use of 3-dimensional (3D) graphics, a lot of things can be visualized, from simple cubes to complex models of real-life objects. The need for an application that visualizes geographical and other data about the Earth in 3D has been expressed as early as 1998 by then vice president of the US Al Gore [1]. He proposed many practical uses for such an application, named "Digital Earth". The proposed uses include education, national border negotiations, redeployment of police force to the most problematic areas, biodiversity preservation, climate change prediction and improvement of agricultural productivity. Because most world maps use the Mercator projection, which deforms the view of the world, the visualization of Earth in 3D can teach the children a lot more about our planet than the conventional 2D maps can. The digital model of Earth, combined with many kinds of data (e.g. historical, geodetic, etc.) could also lead to better

understanding of the planet [2].

Quite a few methods for visualization of Earth and planets have been developed and optimized since the historical Gore speech. During the years 2003 to 2006, a group of Italian computer scientists presented a method for high performance terrain visualization, named BDAM, and two of its improvements, PBDAM and CBDAM [3, 4, 5]. Schneider and Westermann have also proposed a method using nested meshes for high quality terrain rendering with minimal GPU overhead in 2006 [6]. In 2007, Schafhitzel et al. proposed a method for rendering of the planets, including a simulation of the atmosphere, in real-time [7]. A short book on planet visualization was written in 2008 by Østergaard, which discusses the implementation challenges of a digital globe and presents the implementation solution with some optimizations [8]. Cozzi and Ring took that achievement one step further in 2011, by writing an exhaustive book on virtual globe design, along with multiple propositions on how to deal with implementation issues that can occur, and the full C# implementation [9].

This paper is focused on real-time rendering of 3D Earth map. A method for a dynamic visualization is presented and explained, and the efficiency of the method is discussed. The proposed method firstly calculates visualization vertices on the screen, then calculates their UV coordinates. The results of these steps fill the Vertex Buffer Object, which is sent to the graphics card. There, the calculated vertices are processed by the vertex shader, geometry shader, and the fragment shader programs, before the map is drawn on the screen. According to experimental results, the proposed method produces high enough FPS (frames per second) to be used in high performance applications. The proposed method is much simpler and light-weight than the state of the art visualization methods. The floating point precision errors are handled by means of normalizing the coordinates to the visible part of the Earth, while the interpolation errors are processed by the geometry shader on the GPU. This paper also does not concern itself with acquisition of texture data; the implementation behind it is out of scope of this paper.

This paper is organized as follows. The method for visualization is explained in Section 2. The results, produced by the method, namely frames per second (FPS) in relation to the number of triangles rendered, are presented in Section 3. The conclusion is given in Section 4.

2. METHODOLOGY

The rendering procedure, presented in this paper, consists of following steps. Firstly, the visible tiles are calculated. Then, the vertices and their UV coordinates are calculated. The vertex data is sent to the graphics card for drawing. Finally, the vertex, geometry, and fragment shader finalize the calculations, and the result is drawn to the screen. These steps are presented in more detail in the following subsections.

2.1 Vertex generation

The first step in 3D graphics pipeline is the generation of vertices. In many applications that use 3D graphics, some modelling software is used to generate the model, already complete with vertices, UV coordinates, and normals of the object. However, such models are not suitable for high-resolution realistic display applications because of three reasons:

- A model file with sufficient number of vertices for realistic display of the highest quality would be extremely large,
- Rendering of too many vertices results in a very low FPS (frames per second) environment when using mediocre graphics cards, due to insufficient available graphics memory or shader processing cores,
- Older versions of shader programs don't support the use of 64-bit precision, which is required for rendering of the most detailed views of the Earth.

Because our goal was compatibility with some of the older versions of shading language, 64-bit precision could not be used in shader programs. Therefore, in presented application, dynamic vertex generation was used, which also meant that the precision-critical data could not be processed by shader programs on the GPU. Because of that, the FPS (frames per second) in our application is a bit lower than it could have been in an ideal setting.

For the vertex generation, a ray casting algorithm is used. The screen is divided in up to 121 equally spaced points, and from each of those points, a ray is sent out. This produces a grid on the screen with up to 10 rows and columns. The rays' intersections with sphere representing the Earth are calculated. If we have a ray $\vec{r} = (rx, ry, rz)$, sphere with radius R , the camera positioned at $\vec{c} = (cx, cy, cz)$, and the dot product between ray direction and camera position vectors dt , the closest intersection, $\vec{p}$, between ray and the sphere can be calculated by Equation ??.

$$\vec{p} = \vec{c} - (dt + \sqrt{dt^2 + R^2 - |\vec{c}|^2}) \vec{r}. \quad (1)$$

Note that the part under the square root is negative when the ray and the sphere do not intersect. In that case, the ray's direction is adjusted a bit, so that it touches the sphere. The tangent point, $\vec{t}$, is calculated using Equation ??, with intermediate vectors $\vec{a}$ (Equation ??) and $\vec{b}$ (Equation ??)

$$\vec{a} = \vec{c} + \frac{|\vec{c}|^2}{-dt} \vec{r} \quad (2)$$

$$\vec{b} = \frac{\vec{a}}{|\vec{a}|} |\vec{c}| \frac{R}{\sqrt{|\vec{c}|^2 - R^2}} \quad (3)$$

$$\vec{t} = \vec{c} + \sqrt{|\vec{c}|^2 - R^2} \frac{\vec{b} - \vec{c}}{|\vec{b} - \vec{c}|} \quad (4)$$

Finally, the coordinates of the points are normalized.

2.2 UV coordinate calculation

The UV coordinates for texturing are calculated from the vertex' coordinates. If the map tiles use Mercator projection, which distorts the image in the direction of vertical axis (V coordinate), that has to be reversed first [10]. The U coordinate is calculated simply from the normalized vertex position, $p = (x, y, z)$, as in Equation ??.

$$U = \frac{\arctan(\frac{x}{z})}{2\pi} + 0.5, \quad (5)$$

With the help of constant M (Equation 7), the V coordinate is then calculated using Equation ??.

$$V = \frac{\operatorname{arcsinh}[\tan(2M \arcsin[y])]}{2\pi} + 0.5, \quad (6)$$

$$M = \arctan[\sinh(\pi)]. \quad (7)$$

Because shader programs use 32-bit precision, an additional step is required when calculating the UV coordinates. When rendering a close-up view, only a really small part of the Earth is visible, which means the difference between minimum and maximum UV coordinates is really small too. Because of that, the UV coordinates can be normalized to the visible part of the Earth. The minimum and maximum visible row and column of tiles were already calculated before vertex generation step. Instead of having the V value 0 at the south pole and 1 at the north pole, we can adjust it to be 0 at the bottom of the bottommost visible row and 1 to be at the top of the topmost visible row. The U coordinate is also mapped from 0 at the left of leftmost visible column and 1 at the right of rightmost visible column. That way, the precision errors in shader programs are kept in check when rendering the high resolution close-up views of the Earth.

2.3 Vertex shader

The vertex shader program's only task is to transform the vertex position from the world space into screen space by multiplying with the standard model-view-projection matrix. In case the model vertex buffer is used instead of the dynamically calculated vertex buffer, the vertex shader also remaps the UV coordinates to adapt them to the Mercator projection, using Equation ?? and the constant M from Equation 7.

$$V' = 0.5 - \frac{\arcsinh[\tan(2M[V - 0.5])]}{2\pi}. \quad (8)$$

2.4 Geometry shader

Geometry shader program has one of the most important tasks in this whole process. It handles the interpolation errors that happen at the edges of the map. Because OpenGL fragment interpolation is always linear, errors occur when one of the points in the triangle has the U coordinate of 0.1 and another point in the same triangle has the U coordinate of 0.9. In that case, the shader should interpolate from 0.1 down to 0, then jump to 1 and from there continue interpolating down to 0.9. The models circumvent this issue by duplicating vertices at the UV coordinate wrap-arounds, so if we're using a model, the geometry shader can be skipped. The geometry shader program takes in the set of vertices representing a primitive, in our case a triangle. It checks the U coordinate of all 3 vertices, and if the difference between any 2 of them is higher than 0.7, it inserts another vertex between them, to split the triangle into 2 or more triangles, setting the inserted point's U coordinate to 0 for one triangle and 1 for the other. It then outputs all of the resulting triangles to the following stages of the rendering pipeline.

2.5 Fragment shader

The fragment shader program's task is to determine the colour of each fragment. OpenGL allows the shader program to access up to 32 different textures during each draw call. But because a lot of textures converge at the north and south pole, multiple draw calls are needed during each frame. Each of those calls is passed one of the tile textures. The fragment shader program is responsible for mapping of UV coordinates to the texture, and discarding all of the fragments that do not sample from the texture, passed in the current call. If the fragment is not discarded in this draw call, the fragment shader program samples the passed texture to obtain the fragment's colour, which is finally output to the depth testing.

3. RESULTS

In this section, the results of rendering the Earth with the proposed method are presented. The average time needed to process and render a frame was measured, and the average FPS (Frames per second) was calculated. The measurements were done on a personal computer with the following configuration: Intel Core i5-3570K CPU 3.40 GHz, 32 GB of RAM, GeForce GTX 1060 Windforce OC 6GB, running 64 bit operating system Windows 10 Education. The width and height of all the texture units used were 256 pixels. Their total size on disk was 9560 terabytes. The result of rendering three different levels of detail (LOD) of the same area is shown in Figure 1.

During standard rendering, the screen was divided into a grid of 10×10 rectangles, each of which was split into two triangles. FPS calculation was done by lowering the grid's dimensions all the way down to 1 rectangle, and measuring the average time required to render a frame. Using that

Figure 1: The result of rendering the same area in three different LOD-s

information, the average FPS was calculated. The results are shown in Table 1.

Table 1: Average FPS when rendering a pre-set number of triangles

Number of triangles	Average FPS
2	452.90
8	350.62
18	397.89
32	407.09
50	405.80
72	364.40
98	356.11
128	370.66
162	382.86
200	359.95

As seen from the table, though the results vary, the average FPS generally tends to decrease with a higher number of triangles. This can be seen in Figure 2, where the dotted line represents the 6th order of the polynomial trending line.

Figure 2: Graph representing the average FPS in relation to the number of rendered triangles, with the 6th order polynomial trending line (dotted)

4. CONCLUSION

In this paper, a method for real-time visualization of Earth in 3D is presented. It works by dynamically calculating the visualization data and uploading it to the graphics card, where it is processed by the vertex shader, geometry shader, and fragment shader. The experimental results have shown that the method produces a very high FPS (frames per second), which allows for the usage in applications, where high performance is important.

5. REFERENCES

- [1] A. Gore. The digital earth: Understanding our planet in the 21st century. *Australian Surveyor*, 43(2):89–91, June 1998.
- [2] M. F. Goodchild. The use cases of digital earth. *International Journal of Digital Earth*, 1(1):31–42, March 2008.
- [3] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno. Bdam - batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum*, 22(2):505–514, November 2003.
- [4] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, R. Scopigno. Planet-sized batched dynamic adaptive meshes (p-bdam). In *Proceedings of IEEE Visualization*, pages 149–155. IEEE, October 2003.
- [5] E. Gobbetti, F. Marton, P. Cignoni, M. Di Benedetto, F. Ganovelli. C-bdam - compressed batched dynamic adaptive meshes for terrain rendering. *Computer Graphics Forum*, 25(3):333–342, December 2006.
- [6] J. Schneider and R. Westermann. Gpu-friendly high-quality terrain rendering. *Journal of WSCG*, 14(1-3):49–56, February 2006.
- [7] T. Schafhitzel, M. Falk, T. Ertl. Real-time rendering of planets with atmospheres. *Journal of WSCG*, 15(1-3):91–98, January 2007.
- [8] J. Östergaard. *Planet Rendering Using Online High-Resolution Datasets*. Linköping University, Norrköping, Sweden, 2008.
- [9] P. Cozzi and K. Ring. *3D Engine Design for Virtual Globes*. A K Peters, Ltd., Natticks, Massachusetts, USA, 2011.
- [10] D. H. Maling. *Coordinate Systems and Map Projections, Second Edition*. Pergamon Press, Oxford, England, 1992.

University of Primorska Press
www.hippocampus.si
ISBN 978-961-7055-26-9