УДК 004.94

Tatyana Demenkova, Leo Tverdokhlebov

Moscow technological university (MIREA), Moscow, Russia

USE OF GRAPHIC PROCESSOR FOR CREATION OF 3D PANORAMA*

ABSTRACT

This work presents a method of rendering of 360° panoramas based on real-world DEM (Digital Elevation Model) data using graphic processor (GPU). Given geographic coordinates as input this method generates a panorama image and also a distance map with a distance from the point of view to every pixel of the rendered panorama. The details on estimating these distances and the results of accuracy evaluation and error measurements are provided.

KEYWORDS

3D panorama rendering; GPU rendering; GPU z-buffer; estimation of distances up to objects of relief; digital elevation model; geo-information systems.

Деменкова Т.А., Твердохлебов Л.В.

Московский технологический университет (МИРЭА), Москва, Россия

ИСПОЛЬЗОВАНИЕ ГРАФИЧЕСКОГО ПРОЦЕССОРА ДЛЯ ПОСТРОЕНИЯ 3D ПАНОРАМЫ

АННОТАЦИЯ

В статье представлен метод построения 3D панорамы с использованием графического процессора (GPU), основанный на цифровой модели высот. Метод генерирует панорамное изображение и маску дальностей для относительно любой географической координаты. Дана оценка точности предложенной модели в сравнении с эталонной моделью с использованием разработанной программной реализации алгоритма.

КЛЮЧЕВЫЕ СЛОВА

Построение 3D панорамы; построение сцены на графическом процессоре; z-буфер графического процессора; оценка расстояния до объектов рельефа; цифровая модель высот; геоинформационные системы.

INTRODUCTION

With rapid development of digital cartography and GIS (geography information systems) a lot of related techniques and approaches aiming at usage of spatial data emerged. The great contribution was made by the NASA's Shuttle Radar Topography Mission (SRTM) giving the public access to highly detailed geospatial data of the whole surface of our planet. Among all the fields of application of digital elevation model (DEM) data [1] it seems rational to highlight the field of comparison between real-life collected data and the model to be able to produce the new, previously non-existent information and conclusions [2,3].

This article presents an algorithm of estimation of the distance to the terrain relief based on the DEM, which can be used for the generation of 360° panoramas together with depth map for all pixels of the panorama. The comparison of the measurements with the etalon model is provided as well.

PROPOSED METHOD OF ESTIMATION

Rendering of the panorama is possible using the graphical processor (GPU). In particular it is interesting to consider the simultaneous rendering of the panorama together with the depth map. Any modern GPU is equipped with z-buffer, which stores the information about the distance to all the objects of the 3D scene. In order to reconstruct linear depth values from the values obtained from z-buffer one needs to reverse engineer the pipeline [4] which data goes through before it gets into the z-buffer.

Let us start from the basics and follow the pipeline. The OpenGL projection matrix

* **Proceedings of the XI International scientific-practical conference «Modern information technologies and IT-education» (SITITO'2016), Moscow, Russia, November 25 - 26, 2016**

(GL_PROJECTION) looks like:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Multiplied by homogeneous point (x_e, y_e, z_e, w_e) in eye space it gives us this point in clip space:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix},$$

where

$$x_c = \frac{2n}{r-l} x_e + \frac{r+l}{r-l} z_e,$$
$$y_c = \frac{2n}{t-b} y_e + \frac{t+b}{t-b} z_e,$$
$$z_c = \frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n} w_e,$$
$$w_c = -z_e.$$

since one only cares about depth component and taking into account the fact that in eye space we equal to 1 the result is:

$$z_c = \frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n}.$$
$$w_c = -z_e$$

Next operation OpenGL is the division of by its w component (perspective division):

$$z_n = \frac{z_c}{w_c} = \frac{\frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n}}{-z_e} = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z_e}.$$

After the perspective division the value is in the range [-1,1] and before it is finally written to the z-buffer it is scaled to the range [0,1]:

$$z_b = \frac{z_n+1}{2} = \frac{f+n}{2(f-n)} + \frac{fn}{(f-n)z_e} + \frac{1}{2},$$

,

where z_b – is the value obtained from the z-buffer.

Looking at the relation between z_e and z_n one can notice it's a rational function and that the relation is non-linear. Such non-linear values of z-buffer are perfectly suitable for solving the visibility problem. It may even seem natural since the precision is higher at the near plane. But such z-buffer values are completely useless for distance measurements.

So the value from z-buffer should be taken to apply the opposite transformations in order to consequently obtain linear depth values.

First we scale it from the range of [0,1] to [-1,1]:

$$z_n = 2z_b - 1.$$

Then we recover z_e:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}.$$

Multiplied by homogeneous point (x_e, y_e, z_e, w_e) in eye space it gives us this point in clip space:

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x_e \\ y_e \\ z_e \\ w_e \end{pmatrix},$$

where

$$x_c = \frac{2n}{r-l} x_e + \frac{r+l}{r-l} z_e,$$

$$y_c = \frac{2n}{t-b} y_e + \frac{t+b}{t-b} z_e,$$
$$z_c = \frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n} w_e,$$
$$w_c = -z_e,$$

since one only cares about depth component and taking into account the fact that in eye space we equal to 1 the result is:

$$z_c = \frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n},$$
$$w_c = -z_e.$$

Next operation OpenGL is the division of by its w component (perspective division):

$$z_n = \frac{z_c}{w_c} = \frac{\frac{-(f+n)}{f-n} z_e + \frac{-2fn}{f-n}}{-z_e} = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z_e}.$$

After the perspective division the value is in the range [-1,1] and before it is finally written to the z-buffer it is scaled to the range [0,1]:

$$z_b = \frac{z_n+1}{2} = \frac{f+n}{2(f-n)} + \frac{fn}{(f-n)z_e} + \frac{1}{2},$$

where z_b – is the value obtained from the z-buffer.

Looking at the relation between z_e and z_n one can notice it's a rational function and that the relation is non-linear. Such non-linear values of z-buffer are perfectly suitable for solving the visibility problem. It may even seem natural since the precision is higher at the near plane. But such z-buffer values are completely useless for distance measurements.

So the value from z-buffer should be taken to apply the opposite transformations in order to consequently obtain linear depth values.

First we scale it from the range of [0,1] to [-1,1]:

$$z_n = 2z_b - 1.$$

Then we recover z_e:

$$z_n = \frac{f+n}{f-n} + \frac{2fn}{(f-n)z_e},$$
$$\frac{2fn}{(f-n)z_e} = z_n - \frac{f+n}{f-n},$$
$$z_e = \frac{2fn}{(f-n)\left(z_n - \frac{f+n}{f-n}\right)} = \frac{2fn}{z_n(f-n)-f-n} = \frac{-2fn}{f+n-z_n(f-n)}.$$

With all these transformations we eventually obtain the linearized value, where n – distance to the near plane of the frustum, f – distance to the far plane of the frustum:

$$\frac{2fn}{(f-n)z_e} = z_n - \frac{f+n}{f-n},$$
$$z_e = \frac{2fn}{(f-n)\left(z_n - \frac{f+n}{f-n}\right)} = \frac{2fn}{z_n(f-n)-f-n} = \frac{-2fn}{f+n-z_n(f-n)}.$$

With all these transformations we eventually obtain the linearized value, where n – distance to the near plane of the frustum, f – distance to the far plane of the frustum.

IMPLEMENTATION

The whole system is implemented in Java programming language. Java gives us the advantage of using the code of the system in Desktop applications, web-services and mobile devices. The rendering of the 3D scenes is performed through the jME3 game engine library with some core classes rewritten in the way that provides the access to the underlying layer of LWJGL library. Access to the underlying layer is needed to be able to directly receive the data from z-buffer of GPU hardware, such low-level functionality is normally not needed to game and conventional 3D software developers.

jME3 itself uses LWJGL (Lightweight Java Game Library). The library accesses native C code through the Java Native Interface (JNI). LWJGL provides bindings to OpenGL. The library acts as a wrapper over the OpenGL native libraries and provides API similar to the OpenGL APIs of other lower lever languages running directly on hardware (as opposed to Java running inside the Java Virtual Machine) such as C\C++. Platform-specific lwjgl.dll file should be accessible to the Java code. Though it's not a big constraint since libraries for most popular operation systems (Linux/Windows) and platforms (x86/x64/ARM) are available. The first attempt was made to implement the renderer using JOGL OpenGL Java library. Although the 360 degrees panorama rendering was successful, it was not possible to access z-buffer data. It forced us to switch to jME3 and LWJGL libraries. Another advantage of jME3 over the JOGL library is the support of DEM height maps out of the box.

No GUI is provided for the panorama generation functionality since the program is supposed to be integrated to the pipeline of the comprising project. The code can anyway be started through command line. The system can be split into some modules:

1. DEM operating module.

This module reads DEM file and produces the 3D landscape model suitable for renderer. Module takes geographical coordinates representing the centre of the desired area and the desired radius as an input. Since the DEM data is stored in the big amount of files one square degree of latitude\longitude per file the module can sew together terrain data from up to 9 files producing the square terrain.

2. Renderer module.

This module performs rendering of the 3D terrain surface. The usable classes of the module are represented by two different classes: OffscreenRenderer and OnscreenRenderer. OffscreenRenderer provides the 360 degree panorama generation functionality. OnscreenRenderer provides the ability to fly over the 3D landscape in a flight simulator-like way using keyboard and mouse controls. Simple GUI is provided to choose screen resolution and some other rendering parameters. OnscreenRenderer is extremely useful for debug purposes and for detection of anomalies in DEM data.

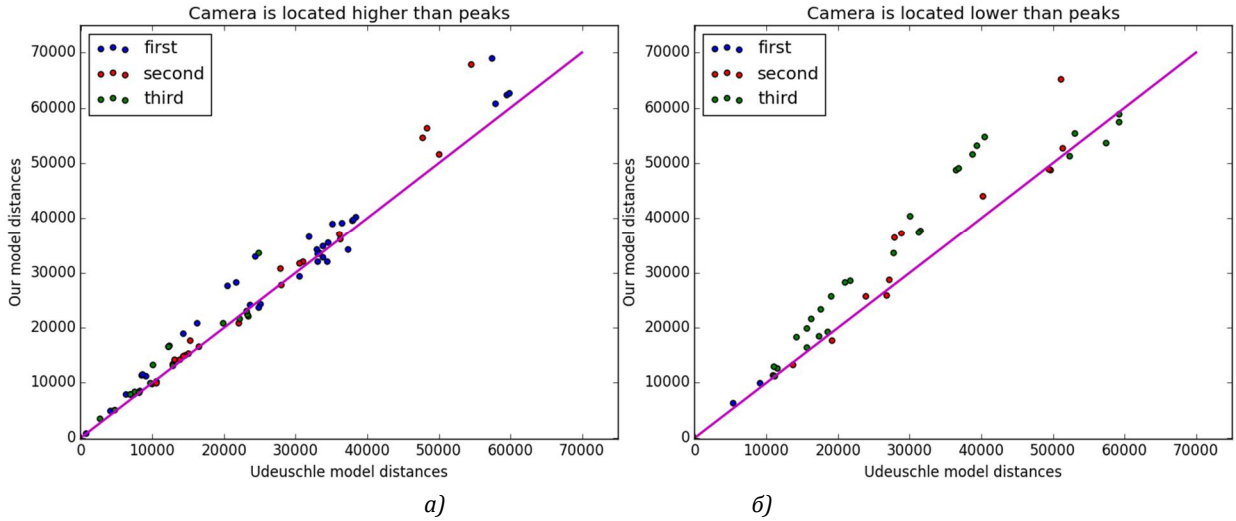
In the beginning a program receives geographical coordinates (latitude and longitude) as an input. Based on the coordinates the set of files containing relevant DEM data is determined and read into the memory. Then these data are concatenated in the proper order to form one continuous terrain surface. After this the terrain is positioned and cut to the requested coordinate the new centre of the terrain surface. Next the DEM heightmap (two-dimensional array of heights in meters) is translated into the 3D model (vertices, edges, faces, and polygons) and the altitude of the landscape in the desired coordinate is determined based on the DEM data. Then camera is positioned into the desired coordinate and on the determined latitude. Initially the camera is facing north with a clockwise shift of

22.5° (half of 45°). Then the camera is rotated by 45° eight times. After each rotation we make a snapshot of what camera sees and a snapshot of z-buffer content. At the end all the data is sewed up together to form 360° view panorama and distance panorama and this data is returned to the calling function to be used in subsequent processing or just be saved as a file for later use.

EXPERIMENTAL EVALUATION OF THE PROPOSED METHOD

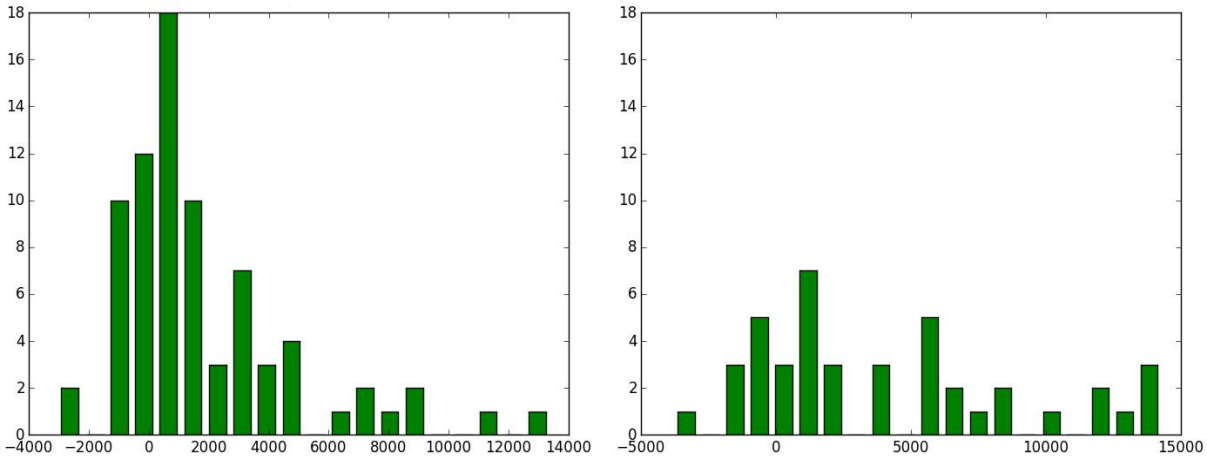
In order to evaluate the accuracy of the distance predictions this work will evaluate the developed model against the mountain view generator of Dr. Ulrich Deuschle (hereinafter referred to as Udeuschle) which we take as a reference model.

First three datasets were collected by comparing three pairs of panoramas and handpicking the mountain peaks which were clearly visible and distinguishable both on picture generated by developed system and picture generated by Udeuschle generator. After plotting of all three datasets on one plot together with the line of perfect correlation it became obvious that all the observations could be naturally divided into two groups. The first group includes the observations made by the camera located lower than the peak, the second group – observations made by the camera located higher than the peak (Figure 1).



а) *б)*

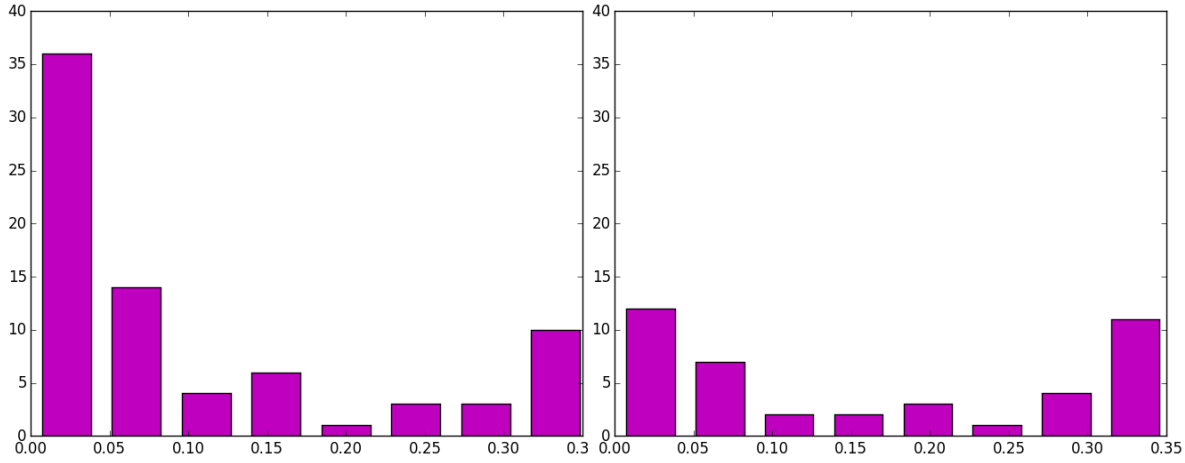
Figure 1. Correlation of distance prediction results between our model and Udeuschle model for cases when: a) camera is located higher than peaks; б) camera is located lower than peaks



а) *б)*

Figure 2. The histogram of the residuals on the evaluation data for cases when: a) camera is located higher than peaks; b) camera is located lower than peaks

As for the residuals, it can be concluded that a residual for an observation in the evaluation data is the difference between the true target and the predicted target. Residuals represent the portion of the target that the model is unable to predict (Figure 2). A positive residual indicates that the model is underestimating the target (the actual target is larger than the predicted target). A negative residual indicates an overestimation (the actual target is smaller than the predicted target). The histogram of the residuals on the evaluation data when distributed in a bell shape and centred at zero indicates that the model makes mistakes in a random manner and does not systematically over or under predict any particular range of target values (the histogram follows the form of the normal distribution).



a) *b)*

Figure 3. Statistical distribution of the absolute percentage error for the cases when: a) camera is higher than peaks; b) camera is lower than peaks

This histogram on the Figure 3 confirms the predictive power of the model. For most of the cases the distance was predicted with the model, as most of the distances are predicted with the percentage error not more than 10% (for most of the points real distance is in the interval [0.9 * predicted value, 1.1* predicted value]).

Table 1. Percentage of distances estimated with particular value of the absolute percentage error

Camera position	<5%	<10%	<15%	<20%	<25%	<30%	<35%	<40%
Higher	51.9%	64.9%	74%	77.9%	83.1%	85.7%	93.5%	100%
Lower	30.9%	47.6%	50%	59.5%	61.9%	71.4%	97.6%	100%
All	44.5%	58.8%	65.5%	71.4%	75.6%	80.7%	95%	100%

This table 1 could be used for the practical applications of our distance predictions model. Imagine one has done an evaluation L of the distance to the peak from the panorama. Based on this value you should

give the shortest interval of the distances, which will include the real distance with probability 80%. If one knows that the peak is higher than the camera, than one could say that the distance to the mountain is L±0.25L with probability 80%. If one has no idea about the height of the peak, he should provide an interval L±0.3L in order to be 80% sure. Now one could try to improve the accuracy of the predictions for the case when the camera is located lower than the peak by calculation of the systematic error.

The dataset was separated into two parts. One part will be used as training set (80%), the second is as test set (20%). The linear regression will be built to find the systematic shift between real and predicted value of the distance. In other words the best coefficients b and c will be found, such that:

dist_real= b * dist_predicted + c + ε,

where:

dist_real – real distance from the peak to the camera,

dist_predicted – distance predicted by our model,

ε – error.

The best b and c are calculated by the least squared estimation method on the training set and the performance are tested on the test set. The best b and c are found using cross validation in order to avoid overfitting. The dataset is divided into k folds (in our case k=7), k-1 folds used as a training set, 1 fold used as a test set. Among this k pair of coefficients the one is chosen, which gives the best value of the R squared statistics. The best values are b=0.898891146468 and c=-1015.66880806.

Table 2. Error metrics with and without linear regression

	Coefficient of determination (R-squared)	RMSE	Relative squared error	Mean Absolute percentage error
Distance predicted by our model	0.831952360592	6360.7378968	0.168047639408	0.166815799976
Distance predicted by our model + linear regression	0.912449916357	4591.1298369	0.087550083643	0.119236588191

Table 3. Percentage of distances estimated with particular value of the absolute percentage error with and without linear regression

Camera position	**<5%**	**<10%**	**<15%**	**<20%**	**<25%**	**<30%**	**<35%**	**<40%**
Higher	51.9%	64.9%	74%	77.9%	83.1%	85.7%	93.5%	100%
Lower	30.9%	47.6%	50%	59.5%	61.9%	71.4%	97.6%	100%
Lower + LR	11.9%	40.5%	66.7%	97.6%	100%	100%	100%	100%

At Figure 4 below one could see the deviation of the predicted value from the etalon value. The closer the point to the line of the perfect correlation the better the model is (according to the perfect model all the point are at this line).

Two consequences could be done directly from the graph. As all the points are quite close to the line of the perfect correlation, our model provides accurate predictions of the distances, the second consequence is the fact that our model is likely to overestimate the distances to the mountain as most of the points are above the line.

So building the linear regression could be a good solution to increase the accuracy of the predictions in cases when it is impossible to install the camera higher than the peaks. But this method could not be used without having some knowledge about real distances to the peak and distances measured by algorithm in order to have enough data to build the regression.

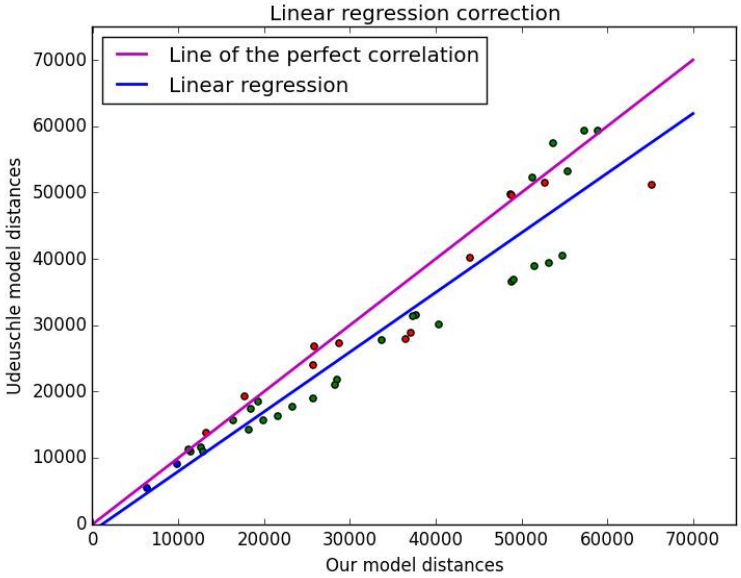


Figure 4. Linear regression correction

CONCLUSION

In this article the method of distance to terrain relief estimation based on digital elevation model was proposed. The specific technique of computing the distance mask based on the transformed data from z-buffer is presented. The experimental results revealed the correlation between the camera-peak altitude gap and the distance error. The evaluation of the accuracy of distance measurements against the reference model is provided. The tables with applied distance accuracy probabilities are computed and directions on usage are given.

The proposed method effectively estimates the distance to the terrain relief. For better precision the measured terrain area should be positioned lower than the camera, in other words it should be directly observable by the camera. This logical requirement is stipulated by the method of the estimation which is based on the data obtained from the z-buffer of the GPU. Yet there is still a great field for improvements in two different areas: rendering optimization and accuracy enhancement.

This work was supported by the Ministry of Education and Science of Russian Federation (project 2014/112 №35).

Литература

1. Farr, T.G., M. Kobrick, Shuttle Radar Topography Mission produces a wealth of data, Amer. Geophys. Union Eos, v. 81, p. 583-585, 2000.
2. M.Hall, D.G. Tragheim. The Accuracy of ASTER Digital Elevation Models, a Comparison to NEXTMap. 2010.
3. SRTM Topography. U.S. Geological Survey, 2003.
4. Fabio Ganovelli, Massimiliano Corsini, Sumanta Pattanaik, Marco Di Benedetto. Introduction to Computer Graphics: A Practical Learning Approach. CRC Press, 2014.

References

1. Farr, T.G., M. Kobrick, Shuttle Radar Topography Mission produces a wealth of data, Amer. Geophys. Union Eos, v. 81, p. 583-585, 2000.
2. M.Hall, D.G. Tragheim. The Accuracy of ASTER Digital Elevation Models, a Comparison to NEXTMap. 2010.
3. SRTM Topography. U.S. Geological Survey, 2003.
4. Fabio Ganovelli, Massimiliano Corsini, Sumanta Pattanaik, Marco Di Benedetto. Introduction to Computer Graphics: A Practical Learning Approach. CRC Press, 2014.

Поступила 16.10.2016

Об авторах:

Деменкова Татьяна Александровна, **доцент кафедры вычислительной техники Московского технологического университета (МИРЭА), кандидат технических наук, demenkova@mirea.ru;**

Твердохлебов Лев Владимирович, **аспирант кафедры вычислительной техники Московского технологического университета (МИРЭА), cosmolev@gmail.com.**