

GRAPHICAL SHELL FOR CONSTRUCTING USER-ENTERED ARITHMETIC FUNCTIONS

Abstract: The article has a relevant topic in the scientific and practical aspect development of a graphical shell of a software application for constructing functions of two variables entered by the user. The choice of the programming language and the use of the OpenGL software interface are justified. The quality of the construction of the framework of the function depending on the calculation step was investigated. A technique for calculating function coordinates for the applied software interface is proposed. The scaling of the test function and the use of GLSL geometry shaders to create lighting simulation are analyzed. The purpose of the work is to create a graphical shell of a software application for constructing functions of two variables entered by the user.

Keywords: function plotting, OpenGL, construction of the frame of the function, recalculation of the coordinates of the function

Description of the problem

Function plotting is useful option for different research areas. In this article plotting of 2-arguments function will be explained. Firstly, it is necessary to define what possible program can be used for.

Main areas of use are visualization of data and function research. Both these areas require to plot some function $f(x, y)$. In the first case value of f is pre-defined in the second case previous computations are required [1, 3].

To build plane that describes function it is necessary to know function $f(x, y)$ and resolution(n) of net.

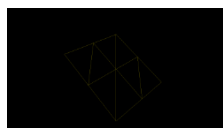
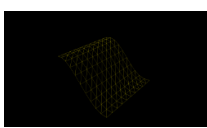
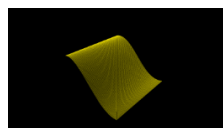
The purpose of the work is to create a graphical shell of a software application for constructing functions of two variables entered by the user

The main content and results of the work

What is meant by resolution is number of cells in net covering particular area of function $f(x, y) = \sin(x)$:

Table 1

Resolution comparison




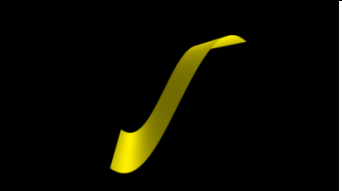

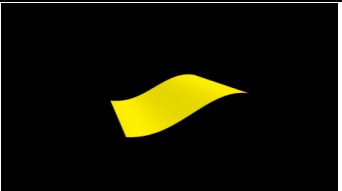
| Resolution | N=2 | N=10 | N=100 |
|------------|---|---|---|
| Result |  |  |  |

General algorithm for net generation for OpenGL use is: divide square with coordinates $[(-1.0,0.0,-1.0), (1.0,0.0,1.0)]$ in $N \cdot N$ pieces; each node of a net will correspond to particular area of function on square $[(x1, y1), (x2, y2)]$; compute value of function f in each node of net; get maximal and minimal values; normalize results using formula:

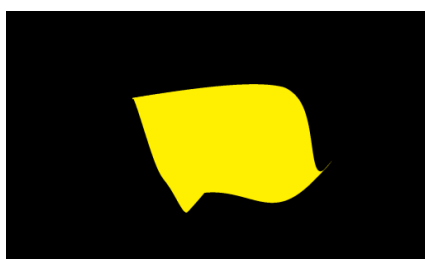
$y_{i,j} = \text{minimal coord} + (\text{maximal coord} - \text{minimal coord}) * (f(x, y) - \text{minimal}) / (\text{maximal} - \text{minimal})$; in this case data will displayed as cuboid with size of 2, $(\text{max coord} - \text{min coord}), 2$.

Table 2

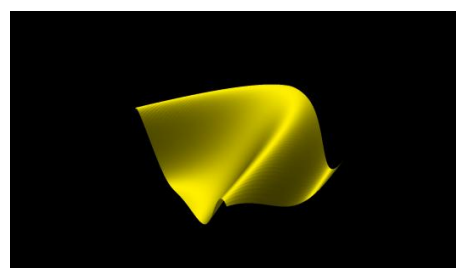
Examples with different values of max cords and min cords

| (min,max) | Top view | Front view |
|--------------|---|--|
| (-0.5,0.5) |  |  |
| (-1.0,1.0) |  |  |
| (-0.25,0.25) |  |  |

Simple plane can't explain completely discrete values of function. On the pictures below two planes can be seen: unlighted drawn with a base color and lighted one [2, 4]. To compare here is example with more complicated function $f(x, y) = \sin(x^2 + y)$.



a) With light



b) Without light

Figure 1. Plotting complicated function $f(x, y) = \sin(x^2 + y)$

To compute “Light” value in geometry shader[plane.frag] next principle is used.

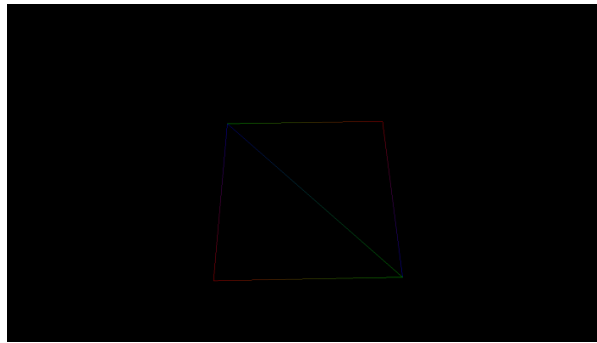


Figure 2. Marked net nodes

On this picture you can see cell of net with vertices, colored according to their number in drawing order:red – first,green-second, blue – third [5]. This vertices gets processed in vertex shader[plane.vert] and resulting triangle primitives are transferred into geometry shader[plane.geom], where light value is computed as absolute dot product between cross product between vectors of borders 0-1 and 0-2 and normal Y-vector 0.0f,1.0,0.0. Then desired plane color gets multiplied by this value and so we get gradation of color from light areas at perpendicular to OY vector triangles and dark areas at parallel.

Also it is useful to know actual function values. To implement such feature two things are required: coordinate system and notes on this system with actual values.

Basic lines of coordinate system can be drawn as three lines corresponding to OX, OY and OZ axis. Attempt to implementation gave following results:

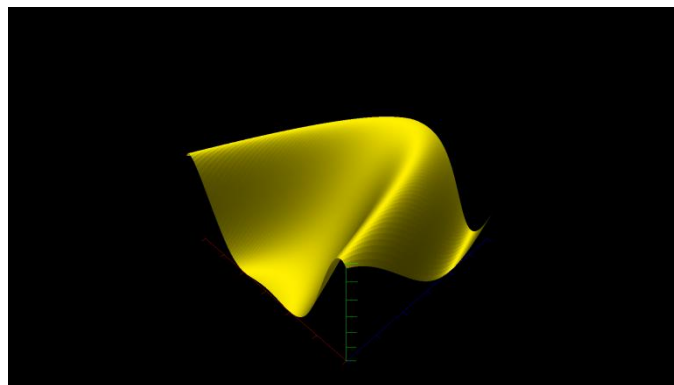


Figure 3. Picture with results

In main.cpp file you can see only main lines of axis given as vector of glm::vec3's named cords. Then this data gets transferred to vertex shader [cords.vert] where we compute position of lines, perpendicular to main axis where in the future notes will be placed. As you can see in corresponding geometry shader [cords.geom] there is field vec4 norm in input block VS_OUT that is used to compute shift vector in function emit_lines that is used to emit extra lines that actually are this perpendicular lines. Value of norm field is computed as coordinates of

top of axis +perpendicular vector of length 0.1 times camera matrix that is computed and transferred to vertex shader in function Matrix of Camera class[camera.h/ camera.cpp] using implemented functions of GLM library that allow us to create view matrix(responds for how objects will be drawn relatively to camera position) and projection matrices (responds for perspective correct drawing and distance, where objects will not be drawn(nearest and most distant)).that is the case for marks on OX and OY axis. In case of the third axis shift vector is equal to 0.1 since we don't want these marks to overlap marks on other axis.

Also there are another fields in block VS_OUT: Axis_number – index of currently drawing axis(used to define value of color vector wince it is useful to have different color for different axis); zero – position of “zero” in coordinate system, from which all computation starts; also there are reflect functions in cords.vert shader that reflect positions of start and end of coordinate systems since we want to see coordinate system on the closes borders of cuboid.

Reflection is computed by following formula:

$$I = 2.0 \cdot \text{dot}(N, I) \cdot N, \tag{1}$$

where I – incident vector(reflected), N – normal vector that specifies reflection axis.

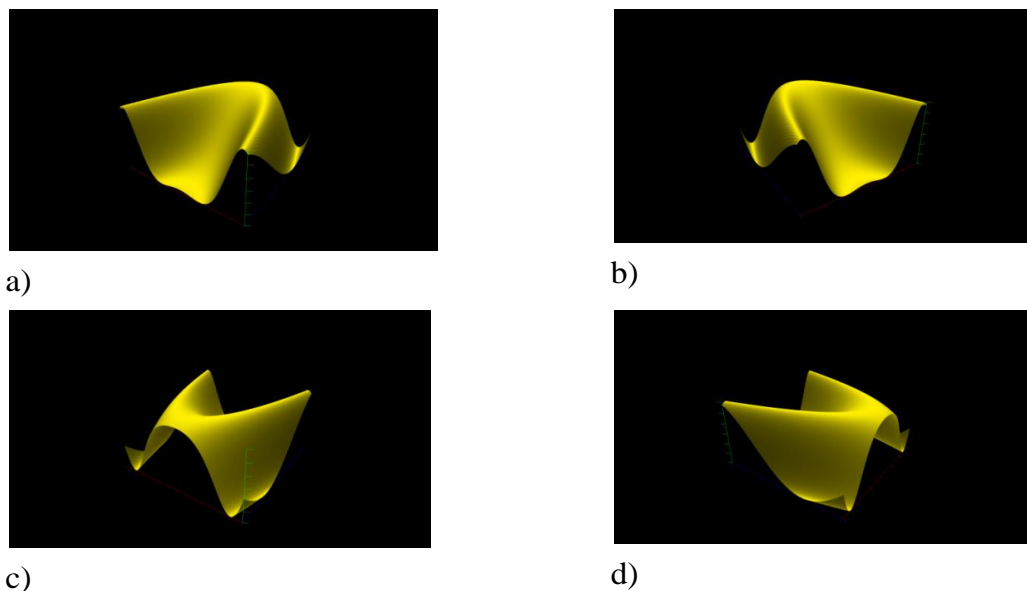


Figure 4. Results with transformations (1) in picture a), b), c), d)

Notes on axes. Since there is no ready functions for text render in OpenGL I decided to implement my own. It is written in files TexLoader.hpp/TexLoader.cpp FontLoader class. As a private members of this class string storedLetters and map object fontMap given. First one is used to store letters that needs to be displayed and map stores pairs char – texture coordinates that allows us to convert number into set of vertices with texture coordinates attribute that can be given to OpenGL to draw a rectangle with attached piece of picture with desired char. In constructor of FontLoader class it is required to give full name of image with special markers – red dots, that mark bottom left and top-right corners of char.

Example of such Image.



Figure 5. Example of a font plate

Two other parameters are texture unit slot that will store image that will be cropped and string with letters that we want to be set as keys of map object fontMap. In constructor image gets loaded as set of bytes using stbi_load function. Then it becomes possible to iterate over rest of pixels and color channels using next picture.

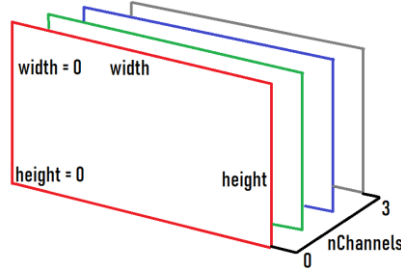
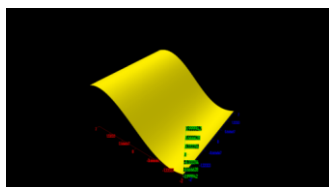
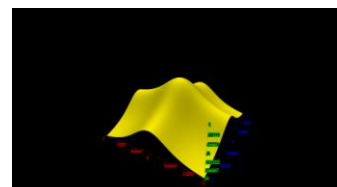


Figure 6. Structure of four channels picture file

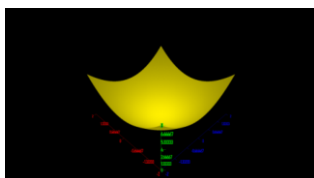
Since stbi_load loads image as 1-dimensional array, position of pixel with coordinates (i, j) can be computed as: $(j \cdot width + i) \cdot nChannels$. Then using this formula program loops over pixels of image, finds pixels with values $(255, 0, 0, *)$ and transfer their positions to corresponding map field according to order given as third argument of constructor. Then, after building class object via constructor we can use functions to_coords and getIndices to get data that can be transferred into Mesh class constructor in order to draw them. Since we have 3 axes with 7 marks we need 21 mesh object with data between minimal and maximal values of corresponding axis and separator equal to $(max-min)/6$.



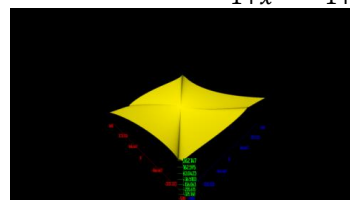
a) $f(x, y) = \sin(x)$



b) $f(x, y) = \frac{1}{1+x^2} + \frac{1}{1+y^2}$

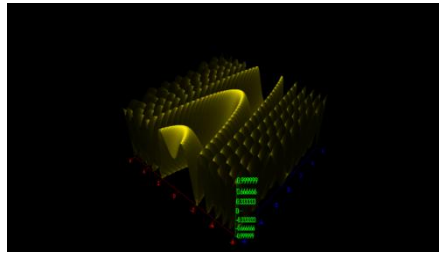


c) $f(x, y) = x^2 + y^2$



d) $f(x, y) = -(y + 47) * \sin(\text{radians}(\sqrt{|y + \frac{x}{2} + 47|})) - x * (\sin(\text{radians}(\sqrt{|x - y - 47|})))$

Figure 7. (Ending on)



e) $f(x, y) = \sin(x \cdot x + y)$

Figure 7. Testing graphical shell with different functions

Conclusion

As a result of work program prototype was made that allows user to display particular area of function with only user input: camera position. Further ways of improvement: allow user to change position of displayed function area, resolution of net and allow to enter custom functions. Created a graphical shell of a software application for constructing functions of two variables entered by the user.

REFERENCES

1. Shkilnyak S. Expressiveness in algebraic systems. Arithmetic predicates, sets, functions. In Academic Council of the Interregional Academy of Management staff (Ed.), Formal models of algorithms and algorithmically calculated functions (4th ed., pp.63). Kiev., Ukraine: Publishing house "Personnel".- 2009.
2. OpenGL 4 Reference Pages. (n.d.). Retrieved October 7, 2022, from <https://registry.khronos.org/OpenGL-Refpages/gl4/>
3. Smoliy V. Management conception designer preproduction of electronic vehicles / V. Smoliy // Адаптивні системи автоматичного управління : міжвідомчий науково-технічний збірник. – 2019. – № 1 (34). – С. 113–124. – Бібліогр.: 22 назви.
4. Core Language (GLSL) - OpenGL Wiki. (n.d.). Retrieved October 7, 2022, from [https://www.khronos.org/opengl/wiki/Core_Language_\(GLSL\)](https://www.khronos.org/opengl/wiki/Core_Language_(GLSL))
5. Victor Gordan. (n.d.). YouTube. Retrieved October 7, 2022, from <https://www.youtube.com/c/VictorGordan>