Computer Graphics (MIEIC)

Practical Work 1

Basic geometry and geometric transformations

Goals

- Install, explore and learn how to use libraries and basic examples for the work and the procedures for the submission of results
- Contact the simplest ways to draw basic geometric shapes using **OpenGL / WebGL**
- Using arrays of geometric transformation to manipulate / modify these geometric shapes
- Use features of **WebCGF**to facilitate the definition and implementation of geometric transformations

Introduction

Is currently possívelgerar interactive 3D graphics in web browsers, use technology**WebGL** and language **JavaScript**. This form of 3D development has the advantages of not requiring the installation of libraries

or compiling applications, and power easily disponibilizaras applications in different operating systems and devices (incuíndo mobile devices). However, for most applications demanding, is recomendávela use of more efficient languages and libraries such as the **C ++** it's the **OpenGL**. However, since the API**WebGL** It is based on**OpenGL**(more specifically**OpenGL ES 2.0**), There are a number of common concepts, whereby**WebGL** It can be seen as a good platform entry in the current versions of the technology **OpenGL**. In the context of **CGRA**We will then resort to**WebGL**and the**JavaScript**to criarpequenas aplicaçõesgráficasque illustrate the conceitosbásicosde Computer Graphics, and allowing the practical experimentation thereof, and may in sercorridas*web browsers*recent**(what support WebGL)**. For apoiaro development, library**WebCGF (Web Computergraphics @ FEUP)**was

developed by teachers of the course and some students, specifically for classes of**CGRA** and**LAIG**In order to abstract some of the complexity of initialization and creation Support features, allowing the focus on components relevant to the concepts of Computer Graphics exploring.

Preparation of the development environment

An important part of this first practical work is the preparation of the development environment. Should to ensure that which has set the main components required, described below, and make sure you can open the browser a sample application.

necessary components

- **Web Browser supporting WebGL**:
  - The application will be effectively performed through the browser. An updated list of browsers what support WebGL can to be found in http://caniuse.com/#feat=webgl. The main browsers for desktop (Google Chrome, Mozilla Firefox, InternetExplorer, Safari) are currently supported, and some mobile browsers like Google Chrome for Android, and iOS Safari (Although in some cases not all devices that can run these browsers have graphics capabilities to run WebGL applications).

- **The basic design structure**:
  - included WebCGF and associated libraries as well as the base folder structure for the project and the HTML file that serves as the base / application entry point.
  - A .zip file containing all the necessary files, including the code base for this practical work is available in Moodle.
  - This structure must be in a folder provided by a web / HTTP server (see next point)

- **HTTP server**:
  - Since the applications accessible via browser, and given the security constraints of mesmosque prevent access to scripts through files on the local disk, It is necessary applications to be made available through an HTTP web server. In the context of CGRA, no dynamic page generation, so any servidorHTTP that delivers static content will. There are several solutions possible for this requirement, including:
    - **Use of FEUP student web area**: Put the project a folder within the public_htmlda folder student account, and accessing via the same public address**http://paginas.fe.up.pt/~eiXXXX/mytest** .In This case, will be poromissão acessívela all (which can be Contoured, one p.ex.com access control file *.htaccess*) .Has Also the disadvantage involve editing / updating files on the server FEUP, and force a connection FEUP network to edit / download application

- **Using a web server on the computer itself**: In some cases students already have a server web (eg Apache, Node.js) to correrpara suportaroutros projetos.O same can be used for this purpose, from the server available the folder with the application through an accessible URL HTTP. If they do not have any server can run one using miniservidor one of the following:
  - **Mongoose web Server** http://cesanta.com/mongoose.shtml (Windows / MAC): A miniservidor web that can be placed in very project folder (or another folder that includes)
  - **Python**: If Python is installed, You can sercriado a server simple HTTP running, the folder that you want to share via the web, O following command (depending on the Python version):
    - `python m SimpleHTTPServer 8080`(For versions 2.x)
    - `python m http.server 8080`(For versions 3.x).

- **A text editor or IDE**: The code that makes up the applications will be written in JavaScript, and stored in text files. For your issue there are several alternatives as well:
  - himself**Google Chrome** available in its**"Developer Tools"(CtrlShiftI)** one debuggerdeJavaScript, which allows passoapasso execution, analysis of variables, the console query, etc.ao code being run in the browser, and allows also map the files accessible by HTTP to the original files stored in disco.Pode that reason they serusado as editore debugger, and is in time **the recommended solution**.
  - Alternatively, the use of an IDE is recommended that JavaScript support, once which can help the error detection syntax and query information related to JavaScript. Eclipse, forexample, has a dedicated package JavaScript in https://eclipse.org/downloads/packages/eclipseidejavascriptwebdevelopers/indigosr2
  - Ultimately, qualquereditorde text can serve However editaros to ficheiros.No, is suggested strongly a publisher text that supports a navigation tree files to allow easy switching between different files that constitute the project.

Test development environment

At this stage you should have a folder with the application / template, shared via a web server. For that reason they should also have the URL address where the folder is acessível.Abra the browser and direcioneo to said URL, and after a few seconds to come up the sample application, can manipulate the view with the mouse, using the left mouse button to rotate the scene, the button right to move sideways and pressing the center button to zoom in / out.

available resources

The library 'WebCGF'

Structure

The library **WebCGF**(Web Computer Graphics @ FEUP) has as main classes as follows:

- ***CGFapplication*(+)**Generate generic issues application startup and libraries support and interconnecting the other components
- ***CGFscene* (*)** It is responsible for initialization, management and design scene
- ***CGFinterface*(*)**It is used to build and manage the interface with the user, may access the internal state of the scene, forexample, ativarou desativarfuncionalidades (p.ex.luzes, animations)

The library also includes the following classes that represent entities that can integrate a scene (non-exhaustive list):

- ***CGFobject*(*)**It is a generic object that the method should deploy***display()*** , the objects to be created must be subclasses ***CGFObject***
- ***CGFlight*(+)**Stores some information associated with a light source (may be extended by subclasses to implement additional features)
- ***CGFcamera* (+)** Stores the information associated with a camera

For the correct execution of the work, it is expected that **may extend the marked classes with(*)**In order to implement the scenes, and interface objects required for each work, as exemplified in the following section. Classes marked with**(+)**are**utility classes example, not exhaustive**To instantiate

parafacilitara management and storage entidadesassociadas (but may be extended by subclasses if they want to add features). The library includes still some predefined objects, as is the case of axles (***CGFaxis***),and more some helper classes, but that should not be necessary for the first practical work the CGRA.

based interaction

In terms of interaction, by default it is possible to manipulate the view using the mouse as follows:

- Left button to rotate the scene around the origin
- Center button (pressed wheel) zoom in / out, or alternatively can be used CTRL + left button
- Right click "slide" the camera sideways

codebase exercise

The basic code provided for the exercise extends the classes mentioned in the previous section in order to implement the design of a very simple scene. The class ***TPscene*** extends ***CGFscene*** And implements methods ***init ()*** and ***display()*** :

- ***init ()***: Contains the code that runs once at the beginning, after the initialization application. Aquique typically initialize variables, create objects or calculations are made Intensive whose results can be stored for later reference.
- ***display()***: Contains the code that actually draws the scene repeatedly. this method will be the focus of this first work.

Please read the comentáriosdisponíveis code in these two methods, they provide important information on its operation and use. In particular, the sample code contained in the method ***display()*** It is divided into three sections:

- Starting the background, camera and axes
- geometric transformations
- primitives Drawing

Estetrabalhofocaráo primitive design and your organization, the declaration and use of geometric transformations, and combining the two to produce a compound geometry.

Practical work

Over the following points are described various tasks to accomplish. Some of them are noted

with the icon  (Image capture) .Nestes points should, capture an image of the application

to drive (p.ex.usando AltPrtScr in Windows or CmdShift3 in Mac OS X to capture for

clipboard and then save to file an image management utility to choose from). At the end of each

class should renomearasimagenspara format **"CGFImagetp1x.y.png"**,on what**x** and**y**

match point and sub-point corresponding to the task (eg **"CGFImagetp12.4.png"**).

The tasks marked with the icon  (Code) must create a .zip file**the folder containing**

your code (typically in the 'tp1 folder, if you are code incluamno other folders

also) And as nomeálo **"CGFCodetp1x.y.zip"**(With x and y identifying this task as

described above). At

the end (Or along the work), should submeteros files via Moodle, using the link

provided for this purpose. They should also include a file**ident.txt** with the elements of list

group (name and number). It will take only one element of the group submitting the work.

1. Basic geometry and structure

The drawing objects in**WebGL** and in modern versions of**OpenGL** It is typically based on

defining a set of triangles with a set of associated characteristics. These triangles are defined by a set of vertices (and possibly some characteristics

associated with each vertex), and how the vertices are connected to form triangles. For example, one considerese rectangle vertices A, B, C and D, comprised of two triangles

vertices ABC and DCB (Figure 1, corresponding code in the file **MyObject.js**).

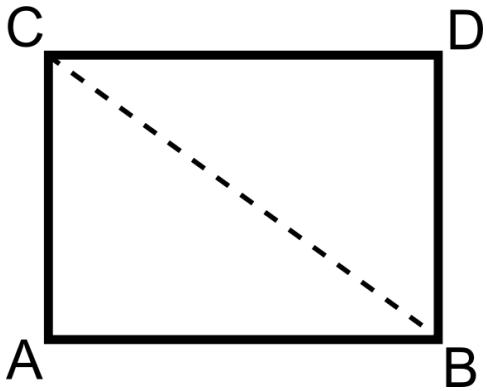

Figure 1: Sample geometry (rectangle).

The basic code supplied with the practical work is provided the code necessary to create a

rectangle, is included in the scene in order to be visible.

For example, a rectangle with corners A, B, C and D may be defined by joining the two triangles

vertices ABC and DCB.Para create this geometry, first create a*array* in

vertices with coordinates of the four corners.

```
= vertices [
        xA, yA, zA, xB,
        yB, zB,

        xC, yC, zc, xD,
        yD, zd

];
```

Then we indicate how these vertices are connected together to form triangles. For this we create a new

array indicating, using indices relating to the order of vertices,

as agrupálos threes. In this case, since the triangles defined by the order ABC and DCB,

we have:

```
indices = [
        0, 1, 2,
        3, 2, 1
];
```

The use of indexes helps reduce the amount of information required to define the geometry. In

Rather than repeat the 3 coordinates vertex list when the same vertex is used more than

once, just repeat its index in the index list. The higher the geometry and the number of shared vertices (something quite common in models

3D composed of a mesh of triangles), there is more benefit in using indices to represent the

connectivity ..

Having this information set, the actual design geometry involves passing the information so

declared**JavaScript**for buffers**WebGL**(Already allocated to the graphics memory), and instruct the same

to draw considering its connectivity as sequences of triangles.

At**WebCGF**The complexity of this finaldo design phase is encapsulated in class**CGFobject**.

Thus, to create a particular 3D object, we can simply:

- create a subclass of **CGFobject**, E.g. **MyObject**
- implement the method **initBuffers**, at where
    - declare the above arrays,
    - invoke the function **initGLBuffers** for information to be passed to the **WebGL**
- In our scene:
    - Create and initialize an instance of the new object in the method *init ()* scene
    - Invoke method **display()**this instance of the object in the method *display()* scene

Exercise

1.Modifique the given object, such that, beyond the already defined rectangle also includes a isosceles triangle with a base unit 2 side, and one height unit, based on the rectangle, leaving half drive margin on each side (centered as the roof a house, see Figure 2). (1.1)

Suggestion : You must change the contents of the list of vertices and indices list.

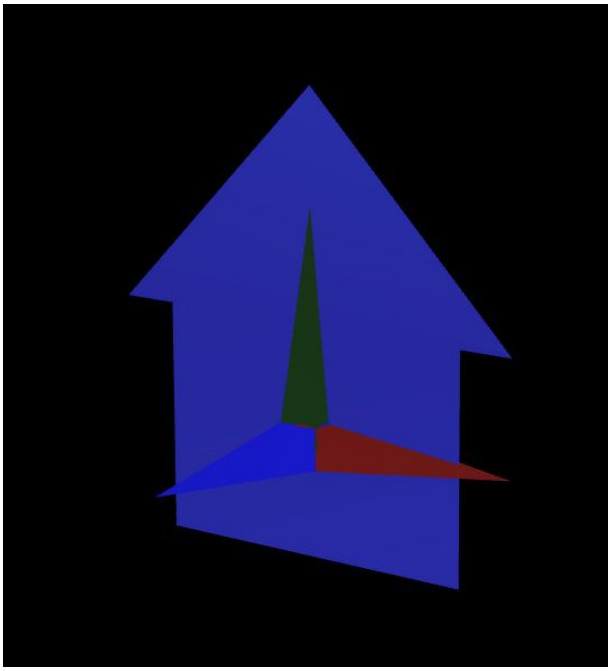

Figure 2: Expected result Ex 1.1..

2. Matrix geometric transformations

A 3D coordinate system, the three basic geometric transformations Translation, Rotation and Escalation is representable by a square matrix with 4 rows and 4 columns. concatenation a set of geometric transformations is obtained by multiplying the respective arrays. In **OpenGL / WebGL** The format of the matrizesde transformaçõesgeométricascorresponde to transposed matrices defined, so the pretenderse a matrix with the following contents:

```
| 0 1 2 3 | | 4
5 6 7 | | 8 9 AB
| | CDEF |
```

it must be done in **OpenGL / WebGL**:

```
m = [
        0, 4, 8, C
        1, 5, 9, D
        2, 6, A, E,
        3, 7, B, F
];
```

The method ***display()*** It contains a section on geometric transformations. In this section find:

- The definition of three arrays of geometric transformation (attention to the comments that are made)
- A concatenation block / multiplication of the previous matrices (in comment, is not serutilizada), where you will find commands like ***this.multMatrix (...)*** , Using the method ***multMatrix*** gives **CGFscene** to accumulate these changes the perspective of the camera, in so that the objects to be processed for the same.

Exercise

1. Descomente each time, one and only one of the matrix multiplication operations corresponding to different transformations, and run the program
2. DescomenteaaplicaçãodaTranslaçãoedaRotação, mantendooescalamentoem comment, and run the program
3. Troque order the two previous changes and run the program;
4. Place again all lines in their order and state initial and repeat the above points, but with Translation and scaling ( 2.4).

3. WebCGF functions for geometric transformations

THE **WebCGF** It provides in its class **CGFscene** a set of instructions for manipulating geometric transformations and apply them to the perspective of the camera, based on library **glmatrix.js**;

with them it is not necessary to declare arrays or converting degree angles to radians. Are they:

- **●CGFscene.*translate (x, y, z)*** : Generates an array of translation and APPLICATIONS;
- **●CGFscene.*rotate (ang, x, y, z)*** : Generates a rotation matrix*ang*radiansaround the axis*(X, y, z)* and APPLICATIONS
- **● CGFscene.*scalc (x, y, z)***: Generates an array of scaling in three directions and APPLICATION; Note: none of the components should be zero, otherwise the geometry will be reduced to something planar, with uncertain effects.

Before performing the following paragraphs, comment entire section of geometric transformations, including instruçõesde concatenation (when viewed, the object formed by the two primitive must be returned to its original position).

Exercise

Each 1.Substitua trêsmultiplicaçõesde arrays by invoking the method of **CGFscene**corresponding to the desired geometric transformation, e.g. ***this.translate (...);*** .

2.Repita points in the previous section, uncommenting and changing the order of instructions recémintroduzidas. At this point must be active scaling and translation.

3.Copie the line that draws the object to before applying the changes. When running program must have two copies of the object, the original position, and the other in position and scale resulting transformations.

now 4.Aplique a 5-units YY translational operation before the copy you created (or You should have the following translation, object, scaling, translation, object). Note that the run the program, both objects were affected by the translation that was added. (

 3.4)

5.Recorrendo the instructions***CGFscene.pushMatrix ()*** and**CGFscene.*popMatrix ()*** , Ensure that the second object is in the same position it was at the end Point 3 (ie only affected by the scaling and initial translation). ( 3.5) ( 3.5).

4. Geometry Unit composed Cube

Until now, only been considered coplanar surfaces. This year the intention is to create

a unit cube, that is, a**cube centered at the origin and unit edge,** ie,

coordinates of (0.5, 0.5, 0.5) and (0.5, 0.5, 0.5). They will be explored two ways to do this: constructing a single mesh of triangles, in talcomo

earlier years or constructing a composite object, wherein the one or more mesh triangles

They are used in the same object (possibly suffering different transformations). Start by making a copy of the project folder for your file, and then delete function

display() all previous code related to the geometric transformations and the two objects of

to take the method***display()*** just draw the axes (ie, remove all the code between the

drawing the axis and the end of the method ***display()***).

Exercise

1. Create a file***MyUnitCube.js*** and set this file class***MyUnitCube*** subclass
   gives***CGFobject***(You must use a copy of the code***MyObject*** as a starting point) .This
   class must define the function***initBuffers*** 8 vertices of the cube, and connectivity between them
   so as to form triangles that constitute the cube square faces. it is recommended that
   Comments are inserted identifying the vertices and faces that are being defined.
2. Deve add the file***main.js*** the inclusion of the new file***MyUnitCube.js*** , On the line where
   the other project files are included.
3. Inclua a new object of type***MyUnitCube*** the function***init*** gives***TPscene*** And invoke the method
   display() in***MyUnitCube*** Full name***display()*** gives***TPScene*** . Run the application. Must have a
   unit cube centered at the origin.

   ( 4.3)
4. Iremos now create a new unit cube, using the design of several square
   unitários.Para this, start by retrieving the file***MyObject, js*** originaldo .zip provided, and
   change its name to***MyQuad.js***And to replace the name of the methods in your
   inside MyObject ... to MyQuad ....
5. De manner analogous to that made in paragraph 1, now create a new file ***MyUnitCubeQuad.js***and
   set in that file class***MyUnitCubeQuad*** as subclass***CGFobject.*** We will not
   necessitarde buffers in**MyUnitCubeQuad**Since the basic geometry is defined in
   MyQuad.remove So the invocation***this.initBuffers ()*** builder, and the function itself
   MyUnitCubeQuad.prototype.initBuffers ().
6. Change then the builder***MyUnitCubeQuad*** in order to add to a member class
   quad which is a new instance of **MyQuad,** and inicializála (note: do not copy the code
   directly from this document may include special characters that create errors in the script).

```
this.quad = new MyQuad (this.scene);
this.quad.initBuffers ();
```

7.Deve now create the function**MyUnitCubeQuad.prototype.display ()**, Which is responsible for desenharas six faces of the cube using ***MyQuad*** previously definido.Nesta function draw the face parallel to the XY, Z = + 0.5, similar to that used in**MyObject**original, and using a translation, must use ***this.scene.translate (...)*** in order to apply the translation in the context of the scene, and ***this.quad.display ()*** to draw the face itself.

8.Desenhe other faces applying the necessary changes, and invoking the drawing *quad* repeatedly.

9.De analogy to point 2, includes a***MyUnitCubeQuad***the function***init*** gives***TPscene***,and invokes its method ***display()***the function***display*** gives***TPscene***.However, In this case **precedao a translation 2 units X**To stand beside the***MyUnitCube*** servant previously. Do not forget to include reference to***MyUnitCubeQuad.js*** and the***MyQuad.js*** at the file ***main.js***

10. When you run the program, must have two unit cubes, one centered at the origin, and another two units next to ( 4.10) ( 4.10)

5. composed Geometry Scene

Now the intention is to implement the code to a more complex scene, comprising umamesa (tampoepernasparalelepipédicas) anda ground (outroparalelepípedo) talcomo shown in Figure 3.

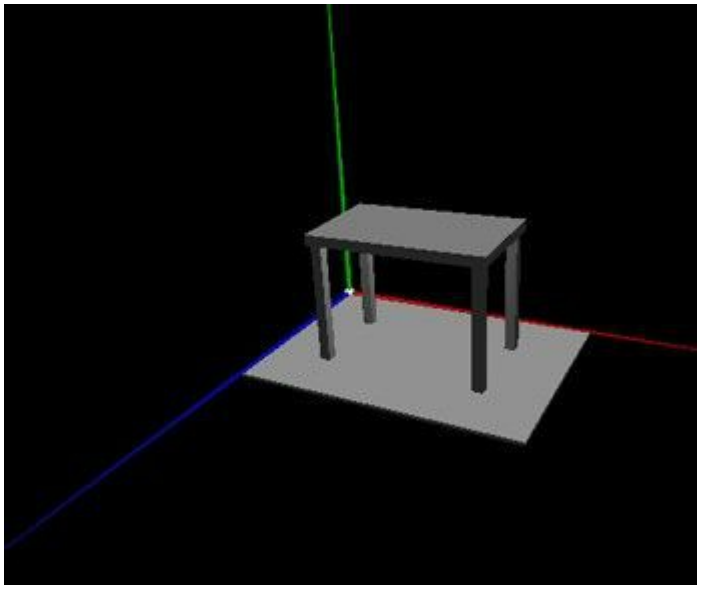

Figure 3: Illustrative scene image to develop.

Exercise

1. Create a new class now ***myTable*** And replace the ***TPscene*** the unit cubes by ***myTable***
2. Acrescente the class***myTable*** an instance of an object***MyUnitCubeQuad*** , And add method***display()***gives***myTable*** to apply transformations and invocations of the method ***display()*** Cube created to create the stand parts **based on the XZ plane**And focused on origem.A table consists of cobblestones: four legs (**dimensions 0.3 * 3.5 * 0.3 units**) And top (**Dimensions 5 * 0.3 * 3 units**).
3. Crie a class***myFloor*** like the previous ones, adding to***TPscene*** an instance the same, and write in the method***display()*** in***myFloor*** the code necessary to draw ground, based on the XZ plane, Also parallelepiped, the dimensions (**dimensions 8 * 6 0.1 * units**).

The method 4.Acrescente***display()*** gives***TPscene*** the changes needed to move the floor and the table so that two edges of the floor coincides with the axes X and Z

(5.4) (5.4)

Check list

Until finaldo work should submeteras following images and versions of code via Moodle,
strictly respecting the rule of names And the file**ident.txt** identifying the
Group members:

-  **Images** (7): 1.1, 2.4, 3.4, 3.5, 4.3, 4.10, 5.4 **(Names like "CGFImagetp1x.y.png"**)
-  **Code zip file**(3): 3.5, 4.10, 5.4 **(Type names "CGFCodetp1x.y.zip")**