My Project

Generated by Doxygen 1.7.6.1

Mon Sep 16 2013 17:21:31

Contents

1 Class Index **1**

1.1 Class List 1

2 Class Documentation **3**

2.1 _GLMgroup Struct Reference 3

2.2 _GLMmaterial Struct Reference 3

2.3 _GLMmodel Struct Reference 4

2.4 _GLMnode Struct Reference 4

2.5 _GLMpolygon Struct Reference 5

2.6 _GLMtriangle Struct Reference 5

2.7 bonus Struct Reference 5

2.8 coordxy Struct Reference 6

2.9 Manager Class Reference 6

2.9.1 Detailed Description 6

2.9.2 Member Function Documentation 6

2.9.2.1 HandleCollision 6

2.9.2.2 UpdateVel 6

2.10 normal Struct Reference 7

2.11 Object Class Reference 7

2.12 Objectrender Class Reference 8

2.12.1 Constructor & Destructor Documentation 8

2.12.1.1 Objectrender 8

2.12.2 Member Function Documentation 8

2.12.2.1 LoadBMP 8

2.12.2.2 LoadImage 9

2.12.2.3 LoadTGA 9

2.12.2.4 Render 9
2.13 terrain Class Reference 9
2.13.1 Detailed Description 10
2.13.2 Constructor & Destructor Documentation 10
2.13.2.1 terrain 10
2.13.3 Member Function Documentation 10
2.13.3.1 getHeight 10
2.13.3.2 Read 11
2.13.3.3 Render 11
2.13.3.4 Render1 11
2.13.4 Member Data Documentation 11
2.13.4.1 ad 12
2.13.4.2 data 12
2.13.4.3 heightmap 12
2.13.4.4 ids 12
2.13.4.5 normals 12
2.13.4.6 pathname 12
2.13.4.7 terrainheight 12
2.13.4.8 terrainwidth 12
2.13.4.9 textures 12
2.14 Texture Class Reference 13
2.14.1 Detailed Description 13
2.14.2 Constructor & Destructor Documentation 13
2.14.2.1 Texture 13
2.14.2.2 Texture 14
2.14.3 Member Function Documentation 14
2.14.3.1 getHeight 14
2.14.3.2 LoadImage 14
2.14.3.3 Render 14
2.14.4 Member Data Documentation 14
2.14.4.1 data 14
2.14.4.2 pic 14
2.15 Vector3D Class Reference 14
2.15.1 Detailed Description 16

2.15.2 Constructor & Destructor Documentation 16
2.15.2.1 Vector3D 16
2.15.2.2 Vector3D 16
2.15.2.3 Vector3D 16
2.15.3 Member Function Documentation 16
2.15.3.1 _cross 16
2.15.3.2 _negate 16
2.15.3.3 _normalize 17
2.15.3.4 cross 17
2.15.3.5 dot 17
2.15.3.6 dot 17
2.15.3.7 mod 17
2.15.3.8 operator∗ 17
2.15.3.9 operator∗= 18
2.15.3.10 operator+ 18
2.15.3.11 operator+= 18
2.15.3.12 operator- 18
2.15.3.13 operator-= 18
2.15.3.14 operator= 18
2.15.3.15 project 18
2.15.3.16 project 19
2.15.3.17 toString 19
2.15.3.18 unit 19
2.15.3.19 unit 19
2.15.4 Member Data Documentation 19
2.15.4.1 z 19

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_GLMgroup 3
_GLMmaterial 3
_GLMmodel 4
_GLMnode 4
_GLMpolygon 5
_GLMtriangle 5
bonus 5
coordxy 6
Manager 6
normal 7
Object 7
Objectrender 8
terrain
    Terrain class 9
Texture
    Texture Class 13
Vector3D
    Vector Class 14

Chapter 2

Class Documentation

2.1 _GLMgroup Struct Reference

Public Attributes

- char ∗ **name**
- GLuint **numtriangles**
- GLuint ∗ **triangles**
- GLuint **material**
- struct _GLMgroup ∗ **next**

The documentation for this struct was generated from the following file:

- glm.h

2.2 _GLMmaterial Struct Reference

Public Attributes

- char ∗ **name**
- GLfloat **diffuse** [4]
- GLfloat **ambient** [4]
- GLfloat **specular** [4]
- GLfloat **emmissive** [4]
- GLfloat **shininess**

The documentation for this struct was generated from the following file:

- glm.h

2.3 _GLMmodel Struct Reference

Public Attributes

- char ∗ **pathname**
- char ∗ **mtllibname**
- GLfloat **translacao** [3]
- GLfloat **escala** [3]
- GLfloat **rotacao** [4]
- GLuint **numvertices**
- GLfloat ∗ **vertices**
- GLuint **numnormals**
- GLfloat ∗ **normals**
- GLuint **numtexcoords**
- GLfloat ∗ **texcoords**
- GLuint **numfacetnorms**
- GLfloat ∗ **facetnorms**
- GLuint **numtriangles**
- GLMtriangle ∗ **triangles**
- GLint **numpolygons**
- GLMpolygon ∗ **polygons**
- char ∗ **texture_file**
- GLuint **nummaterials**
- GLMmaterial ∗ **materials**
- GLuint **numgroups**
- GLMgroup ∗ **groups**
- GLfloat **position** [3]

The documentation for this struct was generated from the following file:

- glm.h

2.4 _GLMnode Struct Reference

Public Attributes

- GLuint **index**
- GLboolean **averaged**
- struct _GLMnode ∗ **next**

The documentation for this struct was generated from the following file:

- glm.cpp

2.5 _GLMpolygon Struct Reference

Public Attributes

- GLuint **numvertices**
- GLuint **n**
- GLuint **t**
- GLuint ∗ **vindices**
- GLuint ∗ **nindices**
- GLuint ∗ **tindices**
- GLuint **findex**

The documentation for this struct was generated from the following file:

- glm.h

2.6 _GLMtriangle Struct Reference

Public Attributes

- GLuint **vindices** [3]
- GLuint **nindices** [3]
- GLuint **tindices** [3]
- GLuint **findex**

The documentation for this struct was generated from the following file:

- glm.h

2.7 bonus Struct Reference

Public Attributes

- Vector3D **position**
- bool **taken**

The documentation for this struct was generated from the following file:

- terrain.h

2.8 coordxy Struct Reference

Public Attributes

- float **x**
- float **y**

The documentation for this struct was generated from the following file:

- 3.cpp

2.9 Manager Class Reference

`#include <physics.h>`

Public Member Functions

- void next ()

  Funtion for updating the position of the bike.
- void UpdateVel ()

  Function for updating the velocity.
- void HandleCollision ()

  Function for handling collisions.

2.9.1 Detailed Description

Physics Class implementing all functions to update velocity, distance etc.

2.9.2 Member Function Documentation

2.9.2.1 void **Manager::HandleCollision ( )**

Function for handling collisions.

This function handles the collisions between the bike and other obstacles placed on the ground.

2.9.2.2 void **Manager::UpdateVel ( )**

Function for updating the velocity.

This function updates the velocity of the bike based upon whether the bike is in air or not, and the terrain around it on which it stands. do your stuff for low velocities.

tilting and speeding of bike

The documentation for this class was generated from the following files:

- physics.h
- physics.cpp

2.10 normal Struct Reference

Public Attributes

- float **x**
- float **y**
- float **z**

The documentation for this struct was generated from the following file:

- terrain.h

2.11 Object Class Reference

Public Member Functions

- void Draw ()

  Draws bike(s), obstacles (trees and crates)
- void addObj ()

  Draws solid cube.
- void **addMarker** ()
- void init (void)

  Makes the Display list of all the objects.

Public Attributes

- Vector3D **position**
- Vector3D **velocity**
- float **size**
- float **vmag**
- float **height**
- float **ctheta**
- float **cphi**
- float **calpha**

The documentation for this class was generated from the following files:

- Object.h
- Object.cpp

2.12 Objectrender Class Reference

Public Member Functions

- GLuint LoadImage (char ∗)

  this function loads the image to be mapped on the object
- void Render (void)

  This function renders the model.
- Objectrender (char ∗, char ∗)

  Constructor.
- GLuint LoadBMP (char ∗)

  this function loads the image to be mapped on the object
- GLuint LoadTGA (char ∗)

  this function loads the image to be mapped on the object

Public Attributes

- char ∗ **objpath**
- char ∗ **filepath**
- GLMmodel ∗ **myModel**
- GLuint **s**
- char ∗ **data**
- int **terrainwidth**
- int **terrainheight**
- Texture **objtex**

2.12.1 Constructor & Destructor Documentation

2.12.1.1 Objectrender::Objectrender (char ∗ *g,* char ∗ *j*)

Constructor.

This function makes on object of Objectrender class /param char∗g This field specifies the obj file /param char∗j This field specifies the texture file

2.12.2 Member Function Documentation

2.12.2.1 GLuint Objectrender::LoadBMP (char ∗ *pic*)

this function loads the image to be mapped on the object

this function load image according bmp format and calls the corresponding function /param char∗ pic The path of th eimage to be mapped

2.12.2.2 GLuint **Objectrender::LoadImage** (char ∗ *pic*)

this function loads the image to be mapped on the object

this function load image according to format either tga or bmp and calls the corresponding function /param char∗ pic The path of th eimage to be mapped

2.12.2.3 GLuint **Objectrender::LoadTGA** (char ∗ *pic*)

this function loads the image to be mapped on the object

this function load image according tga format and calls the corresponding function /param char∗ pic The path of th eimage to be mapped they retrun the id of the texture

2.12.2.4 void **Objectrender::Render** (void)

This function renders the model.

this function invokes the glmdraw function to render the object

The documentation for this class was generated from the following files:

- obj1.h
- obj1.cpp

2.13 terrain Class Reference

Terrain class.

`#include <terrain.h>`

Public Member Functions

- void Render (void)

  Renderer on Terrain with texture.
- void Render1 (Texture ∗)

  renders the terrain
- void Read (void)

  Reads the texture images and heightmap.
- Texture ∗ loader ()

  this function loads the texture image and returns the texture object
- terrain (char ∗pic)

  This is the constructor of the terrain class.
- terrain (void)

  Null constructor of terrain class.

- GLfloat getHeight (int, int)

  Computes height at any point in the terrain.

Public Attributes

- const char ∗ heightmap
- GLubyte ∗ data
- int terrainwidth
- int terrainheight
- string textures [4]
- GLuint ids [3]
- Texture ad [4]
- normal ∗∗ normals
- char ∗ pathname
- bonus **markers** [300]

2.13.1 Detailed Description

Terrain class.

This class takes input as heightmap and textures images which are to be mapped on it and on the HUD. It also takes a pathmap to define the track and place objects accordingly.

2.13.2 Constructor & Destructor Documentation

2.13.2.1 terrain::terrain (char ∗ *pic*)

This is the constructor of the terrain class.

This function is an constructor of the terrain class and takes an arguement the name of the heightmap

Parameters

pic	a char array pointer conating heightmap

2.13.3 Member Function Documentation

2.13.3.1 GLfloat terrain::getHeight (int *i*, int *j*)

Computes height at any point in the terrain.

returns the height at a point in heightmap

This function computes height at any point in the heightmap using data field of the terrain this function return the height at a point in terrain using data field of the terrain object /param int i x value of the point whose height is to be calculated /param int j z value of the point whose height is to be calculated

2.13.3.2 void **terrain::Read** (void)

Reads the texture images and heightmap.

Reads the heightmap.

This function is called in init and it makes textures objects and reads the heightmap and pathmap It takes parameters from the object it is called and various parameters associated with a track are assigned in the init function.

This function reads the texture images specified in the terrain object and reads the heightmap. It fills the texture array with textures

2.13.3.3 void **terrain::Render** (void)

Renderer on Terrain with texture.

/ This array stores the location of the markers on the track

This function calls Render1 funciton which actually renders the terrain using frustum culling. Actually firstly this function used to call displaylist of terrain rendering but afterwards frustum culling was used.

Parameters

it	taks no arguement

2.13.3.4 void **terrain::Render1** (**Texture** ∗)

renders the terrain

this function executes the commands of rendering the terrain. it uses frustum culling to selectively render triangles of the terrain.

Parameters

Texture	∗ -A pointer to array of textures

This function renders the terrain on the heightmap and applying the textures specified on the scaled heightmap. /param Texture a[] This stores the textures in the an array which are chosen to be applied accordingly

2.13.4 Member Data Documentation

2.13.4.1 Texture terrain::ad[4]

This array contains the processed texture data returned by the method in texture class and it is used to apply textures on different objects.

2.13.4.2 GLubyte∗ terrain::data

This byte array stores the data of the image file in byte format and is used to determine the height of the terrain and in physics engine

2.13.4.3 const char∗ terrain::heightmap

The filename of the heightmap is stored in this field

2.13.4.4 GLuint terrain::ids[3]

This array stores the ids of the textures which have been genrated by the texture class.

2.13.4.5 normal∗∗ terrain::normals

This is a poinetr to an array of normals pointers storing normals at each point

2.13.4.6 char∗ terrain::pathname

this character array stores the name of the file containing description of the path

2.13.4.7 int terrain::terrainheight

This field stores height of the terrain

2.13.4.8 int terrain::terrainwidth

This field stores the width of the terrain

2.13.4.9 string terrain::textures[4]

This array stores the name of the texture files which are to be mapped on the terrain and certain parts of HUD

The documentation for this class was generated from the following files:

- terrain.h
- terrain.cpp

2.14 Texture Class Reference

Texture Class.

```
#include <first.h>
```

Public Member Functions

- GLuint LoadImage ()

  Loads the Texture.
- void Render (Texture b)

  Maps texture on a surface.
- Texture (const char ∗d)

  Constructor.
- Texture ()

  Constructor.
- GLfloat getHeight (int, int)

  Returns height at a point on the texture.

Public Attributes

- const char ∗ pic
- GLubyte ∗ data
- int **terrainwidth**
- int **terrainheight**
- GLuint **Terrainid**

2.14.1 Detailed Description

Texture Class.

This class takes texture map as input and gives loaded texture map as output

2.14.2 Constructor & Destructor Documentation

2.14.2.1 Texture::Texture (const char ∗ *d*)

Constructor.

This constructs an object of texture class and assigns the image in pic field. /param const char∗ d It contains the filename of theimage to be mapped

2.14.2.2 Texture::Texture ( )

Constructor.

This constructs an object of texture class having null values of all fields

2.14.3 Member Function Documentation

2.14.3.1 float Texture::getHeight (int *x*, int *y*)

Returns height at a point on the texture.

This function was made fore some testing purposes no significance;

2.14.3.2 GLuint Texture::LoadImage ( )

Loads the Texture.

This function loads the given image file into texture format into byte array RGB format.

2.14.3.3 void Texture::Render (Texture *b*)

Maps texture on a surface.

This function maps the texture on a the smae image treating it as a heightmap /param Texture b a texture object which is to be mapped

2.14.4 Member Data Documentation

2.14.4.1 GLubyte* Texture::data

This field has the file loaded in the format RGB in a byte array

2.14.4.2 const char* Texture::pic

This field has the filename of the image to be mapped

The documentation for this class was generated from the following files:

- first.h
- first.cpp

2.15 Vector3D Class Reference

Vector Class.

`#include <Vector3D.h>`

Public Member Functions

- Vector3D ()

  Constructor.
- Vector3D (float a, float b, float c)

  Constructor.
- Vector3D (const Vector3D &v)

  Constructor.
- Vector3D operator+ (Vector3D v)

  Adding two vectors.
- Vector3D operator- (Vector3D v)

  Subtracting two vectors.
- Vector3D operator∗ (float w)

  Multiplying two vectors.
- Vector3D operator∗= (float w)

  Multiplying two vectors.
- Vector3D operator= (Vector3D v)

  Assigning value to a vector.
- Vector3D operator+= (Vector3D v)

  Adding value to a vector.
- Vector3D operator-= (Vector3D v)

  Subtracting value to a vector.
- void _negate ()

  Negating a vector3D.
- float dot (Vector3D v, Vector3D w)

  Dot product of two vectors.
- float dot (Vector3D v)

  Dot product of two vectors.
- float mod ()

  Modulus of a vector.
- Vector3D _normalize ()

  Unitise a vector.
- Vector3D unit (Vector3D v)

  Unitise a vector.
- Vector3D unit ()

  Unitise a vector.
- Vector3D cross (Vector3D v, Vector3D w)

  Cross Product of Vectors.
- Vector3D _cross (Vector3D w)

  Cross Product of Vectors.
- Vector3D project (Vector3D w)

  Projection of Vector.
- Vector3D project (Vector3D v, Vector3D w)

  Projection of Vector.
- void toString ()

  Printing the vector.

Public Attributes

- float **x**
- float **y**
- float z

2.15.1 Detailed Description

Vector Class.

This class implements Vectors in 3D and all the function associated with vectors in 3D

2.15.2 Constructor & Destructor Documentation

2.15.2.1 Vector3D::Vector3D ( )

Constructor.

Null constructor creates a zero vector

2.15.2.2 Vector3D::Vector3D (float *a*, float *b*, float *c*)

Constructor.

This constructor returns a vector whose x ,y ,z components are specified /param float x This is x component of vector /param float y This is y component of vector /param float z This is z component of vector

2.15.2.3 Vector3D::Vector3D (const Vector3D & *v*)

Constructor.

This constructor creates a copy of the vector /param const Vector3d &v It is a vector of VEctor::D type

2.15.3 Member Function Documentation

2.15.3.1 Vector3D Vector3D::_cross (Vector3D *w*)

Cross Product of Vectors.

This function returns the vector which is cross product of two given vectors /param v cross product of v and vector3D object through which it is called is returned

2.15.3.2 void Vector3D::_negate ( )

Negating a vector3D.

this function negates the values of the given vector3D object /param void it negates the vector3D through which the function is called

2.15.3.3 Vector3D Vector3D::_normalize ( )

Unitise a vector.

this function changes the vector into unit vector along its direction /param void Unit vector of vector through which it is called is returned

2.15.3.4 Vector3D Vector3D::cross (Vector3D *v*, Vector3D *w*)

Cross Product of Vectors.

This function returns the vector which is cross product of two given vectors /param v,w cross product of v and w is returned

2.15.3.5 float Vector3D::dot (Vector3D *v*, Vector3D *w*)

Dot product of two vectors.

this function takes the dot product of two vectors and return the value as a float /param v,w Vector3D Objects whose dot products is to be taken

2.15.3.6 float Vector3D::dot (Vector3D *v*)

Dot product of two vectors.

this function takes the dot product of two vectors and return the value as a float /param v Vector3D Objects whose dot products is to be taken

2.15.3.7 float Vector3D::mod ( )

Modulus of a vector.

this function gives the dot product vectors and return the value as a float /param void Modulus of vector through which it is called is returned

2.15.3.8 Vector3D Vector3D::operator∗ (float *w*)

Multiplying two vectors.

This function overloads the ∗ oprator for Vector3D class. /param float w this is a float which is to be multiplied to given Vector3D object

2.15.3.9 Vector3D Vector3D::operator∗= (float *w*)

Multiplying two vectors.

This function overloads the ∗= oprator for Vector3D class. /param float w this is a float which is to be multiplied to given Vector3D object

2.15.3.10 Vector3D Vector3D::operator+ (Vector3D *v*)

Adding two vectors.

This vector defines + for the Vector 3D objects and adds the vector specifed /param Vector3D v this is a vector 3D object which is to be added to a given object

2.15.3.11 Vector3D Vector3D::operator+= (Vector3D *v*)

Adding value to a vector.

This function overloads the += operator for Vector3D class. /param Vector3D v this is a Vector3D object which is to be added to the given vector3D Object

2.15.3.12 Vector3D Vector3D::operator- (Vector3D *v*)

Subtracting two vectors.

This function overloads the - oprator for Vector3D class. /param Vector3D v this is a vector 3D object which is to be subtracted from a given object

2.15.3.13 Vector3D Vector3D::operator-= (Vector3D *v*)

Subtracting value to a vector.

This function overloads the -= operator for Vector3D class. /param Vector3D v this is a Vector3D object which is to be added to the given vector3D Object

2.15.3.14 Vector3D Vector3D::operator= (Vector3D *v*)

Assigning value to a vector.

This function overloads the = operator for Vector3D class. /param Vector3D v this is a Vector3D object which is to be assigned

2.15.3.15 Vector3D Vector3D::project (Vector3D *w*)

Projection of Vector.

This function returns the projection of a vector in the direction given /param w Vector3D object in the direction of which projection is to be taken

2.15.3.16 Vector3D Vector3D::project (Vector3D *v*, Vector3D *w*)

Projection of Vector.

This function returns the projection of a vector in the direction given /param v,w Vector3D object in the direction of which projection of v is to be taken

2.15.3.17 void Vector3D::toString ( )

Printing the vector.

This function was used to print the x,y and z parameters of the vector

2.15.3.18 Vector3D Vector3D::unit (Vector3D *v*)

Unitise a vector.

this function returns unit vector in the direction of given vector /param v Unit vector of v is returned

2.15.3.19 Vector3D Vector3D::unit ( )

Unitise a vector.

this function returns unit vector in the direction of given vector /param void Unit vector is returned through which it is called

2.15.4 Member Data Documentation

2.15.4.1 float Vector3D::z

x,y,z store the x,y,z components of the vector

/∗Constructors: Vector3D v: x, y, and z are all set to 0. Vector3D v2(1, 2, 3): x, y, and z are set to 1, 2, and 3, respectivly. Vector3D v3(Vector3D v4): v3.x, v3.y, and v3.z are all set to v4.x, v4.y, and v4.z, respectivly.

The documentation for this class was generated from the following files:

- Vector3D.h
- Vector3D.cpp