HyperGrafx

31337

Generated by Doxygen 1.8.3.1

Fri Mar 15 2013 00:47:40

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

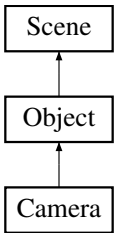
Class Documentation

4.1 Camera Class Reference

The Camera class represents a logical camera in a model view, which posesses a current viewing angle and an absolute position in space as its state.

`#include <Camera.hpp>`

Inheritance diagram for Camera:



Public Types

- enum Direction {
  DIR_FORWARD, **DIR_BACKWARD**, **DIR_LEFT**, **DIR_RIGHT**,
  DIR_UP, **DIR_DOWN**, **DIR_END**, **DIR_BEGIN** = DIR_FORWARD }

  The Direction enumeration lists all of the possible directions the camera may travel in.
- enum ViewType {
  PERSPECTIVE, **ORTHO**, **ORTHO2D**, **IDENTITY**,
  FRUSTUM }

  The ViewType enumeration lists the various possibilities for the current viewing mode that can be switched between.
- enum Uniforms {
  BEGIN = Object::END, **TRANSLATION** = BEGIN, **ROTATION**, **VIEW**,
  CTM, **END** }

  The glsl_var enumeration lists the various variables the Camera class is capable of sending to the shader.
- typedef enum Camera::Direction Direction

  The Direction enumeration lists all of the possible directions the camera may travel in.
- typedef enum Camera::ViewType ViewType

  The ViewType enumeration lists the various possibilities for the current viewing mode that can be switched between.
- typedef enum Camera::Uniforms Uniform

  The glsl_var enumeration lists the various variables the Camera class is capable of sending to the shader.
- typedef const unsigned int **UniformEnum**

- typedef std::map
  < Object::UniformEnum,
  std::string > **UniformMap**

Public Member Functions

- Camera (const std::string &name, GLuint gShader, float x=0.0, float y=0.0, float z=0.0)

  Initialization Constructor; sets the x,y,z coordinates explicitly.
- Camera (const std::string &name, GLuint gShader, vec3 &in)

  Initialization Constructor, uses a vec3 as its initial coordinates.
- Camera (const std::string &name, GLuint gShader, vec4 &in)

  Initialization Constructor, uses a vec4 as its initial coordinates.
- virtual ~Camera (void)

  Default destructor.
- void x (const float &in, const bool &update=true)

  Sets the x coordinate of the camera.
- void y (const float &in, const bool &update=true)

  Sets the y coordinate of the camera.
- void z (const float &in, const bool &update=true)

  Sets the z coordinate of the camera.
- void pos (const float &x, const float &y, const float &z, const bool &update=true)

  Sets the absolute position of the camera.
- void pos (const vec3 &in, const bool &update=true)

  Sets the absolute position of the camera.
- void pos (const vec4 &in, const bool &update=true)

  Sets the absolute position of the camera.
- void dX (const float &by, const bool &update=true)

  Moves the camera along the x axis.
- void dY (const float &by, const bool &update=true)

  Moves the camera along the y axis.
- void dZ (const float &by, const bool &update=true)

  Moves the camera along the z axis.
- void dPos (const float &x, const float &y, const float &z)

  Moves the camera along the x, y, and z axes.
- void dPos (const vec3 &by)

  Moves the camera along the x, y, and z axes.
- void dPos (const vec4 &by)

  Moves the camera along the x, y, and z axes.
- void fieldOfView (const float &fovy)

  fieldOfView sets the current camera Field-of-view angle.
- float fieldOfView (void) const

  fieldOfView() gets the current camera Field-of-view angle.
- void adjustFieldOfView (const float &by)

  adjustFieldOfView adjusts the field of view angle up or down by an amount.
- void changePerspective (const ViewType &vType)

  changePerspective changes the current perspective of the camera.
- void refreshPerspective (void)

  refreshPerspective re-generates the current view/perspective matrix of the camera.
- void viewport (size_t _X, size_t _Y, size_t _width, size_t _height)

  viewport instructs this camera what his expected drawing window will be.
- void sway (const float &by)

Adjusts the camera's x coordinate relative to its current position.

- void surge (const float &by)

  Adjusts the camera's z coordinate relative to its current position.
- void heave (const float &by)

  Adjusts the camera's y coordinate relative to its current position.
- void pitch (const float &by, const bool &fixed=false)

  pitch adjusts the x axis rotation; up/down look.
- void yaw (const float &by, const bool &fixed=false)

  yaw adjusts the y axis rotation; left/right look.
- void roll (const float &by, const bool &fixed=false)

  roll adjusts the z axis rotation; tilt or lean left/right.
- void move (const Camera::Direction &Dir)

  move instructs the camera to begin moving in the specified direction.
- void stop (const Camera::Direction &Dir)

  stop instructs the camera to stop moving in the specified direction.
- void idle (void)

  idle moves the camera forward in whichever directions it is configured to move in.
- void accel (const vec3 &accel)

  accel takes an input vec2 which represents an acceleration, and applies it to the motion vectors with regards to the maximum acceleration and the maximum speed of the camera.
- float x (void) const

  x() returns the current position of the camera in model coordinates.
- float y (void) const

  y() returns the current position of the camera in model coordinates.
- float z (void) const

  z() returns the current position of the camera in model coordinates.
- vec4 pos (void) const

  pos() gets the current camera position in model coordinates.
- virtual void send (Object::UniformEnum which)

  send will send a glsl variable to the shader.
- void view (void)

  view will instruct OpenGL of the viewport we want, and then send all of our current matrices to the shader for rendering.
- void resetRotation (void)

  resetRotation adjusts the camera's rotational state back to its default state (The Identity Matrix.)
- void **Draw** (void)
- void **Buffer** (void)
- void **BufferMorphOnly** (void)
- void **Mode** (GLenum new_node)
- void **Texture** (const char ∗∗filename)
- const std::string & **Name** (void) const
- virtual void **Link** (UniformEnum which, const std::string &name)
- virtual GLuint Shader (void)

  Returns the Object's current Shader.
- virtual void Shader (GLuint newShader)

  Sets the shader to be used by this object.
- void **Animation** (void(∗anim_func)(TransCache &arg))
- void **Propagate** (void)
- vec4 GetPosition () const

  returns the position of the object this makes the lighting implementation much easier...
- Object ∗ **getMorphTargetPtr** () const
- Object ∗ **genMorphTarget** (GLuint)

- float **getMorphPercentage** () const
- void **setMorphPercentage** (const float)
- void **destroyMorphTarget** ()
- int **getNumberPoints** ()
- Object * **AddObject** (const std::string &objName, GLuint Object_Shader=0)
- void **DelObject** (const std::string &objName)
- void **DelObject** (void)
- void **PopObject** (void)
- void DestroyObject (void)

  Completely remove this object and all his children.
- Object * **next** (void)
- Object * **prev** (void)
- Object * **active** (void) const
- Object * **operator[]** (const std::string &objname)

Public Attributes

- std::vector< Angel::vec4 > **points**
- std::vector< Angel::vec3 > **normals**
- std::vector< unsigned int > **indices**
- std::vector< Angel::vec4 > **colors**
- std::vector< Angel::vec2 > **texcoords**
- TransCache **trans**

Protected Member Functions

- void DeleteObject (Object *obj)

  DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.
- void **InsertObject** (const std::string name, Object *obj)

Protected Attributes

- std::string name

  name is used as an identifying handle for the object.
- GLuint vao

  Vertex Array Object handle identifying our buffers/object.
- GLuint buffer [NUM_BUFFERS]

  Handles to our buffers (Vertices, TexUVs, etc.)
- GLenum draw_mode

  Drawing mode for this object.
- bool isTextured

  Is this object textured?
- float morphPercentage

  Morphing/Tweening Things.
- Object * **morphTarget**
- std::map< Object::UniformEnum, std::string > **_uniformMap**
- std::vector< GLint > handles

  Handles to Uniforms on the shader.
- std::list< Object * > **_list**
- std::map< std::string, Object * > **_map**
- std::list< Object * >::iterator **_currentObj**
- GLuint **_gShader**

Private Member Functions

- void adjustRotation (const mat4 &adjustment, const bool &fixed=false)

  adjustRotation is an internal function that rotates the camera.
- void commonInit (void)

  commonInit is a private function that initializes local object attributes.

Private Attributes

- mat4 _view

  The current view matrix (defaultly perspective) for this camera.
- TransCache _ctm

  The Current Transformation state for this Camera.
- ViewType _currentView

  The current viewing mode type.
- GLfloat _speed

  Current Speed of camera motion.
- vec3 _velocity

  Current Velocity of camera motion.
- GLfloat _speed_cap

  Current Speed Capacity: (speed/MaxSpeed)
- GLfloat _maxAccel

  Maximum Acceleration Magnitude.
- GLfloat _maxSpeed

  Maximum Speed.
- GLfloat _frictionMagnitude

  Friction.
- GLfloat _aspectRatio

  Current aspect ratio for certain perspectives.
- GLfloat _fovy

  Current field-of-view angle for perspective view.
- Angel::vec2 _viewportSize

  Camera's Drawbox Width and Height.
- Angel::vec2 _viewportPosition

  Camera's Drawbox x,y Coordinate (Upper-Left Pixel)
- bool _motion [Camera::DIR_END]

  Booleans correlating to the different motion directions.

4.1.1 Detailed Description

The Camera class represents a logical camera in a model view, which posesses a current viewing angle and an absolute position in space as its state.

Author

John Huston, jhuston@cs.uml.edu

Since

16 Nov 2012

Functions are provided to adjust the rotation according to pitch(), yaw() and roll() motions; surge(), sway(), and heave() are provided to adjust position in space.

move(), stop(), and idle() are provided to help the camera automatically move along the x, y, or z axes.

Definition at line 37 of file Camera.hpp.

4.1.2 Member Typedef Documentation

4.1.2.1 typedef enum Camera::Direction Camera::Direction

The Direction enumeration lists all of the possible directions the camera may travel in.

'BEGIN' and 'END' are special sentinel directions for the purposes of iteration, and are ignored by any functions that accept a Direction.

4.1.2.2 typedef enum Camera::Uniforms Camera::Uniform

The glsl_var enumeration lists the various variables the Camera class is capable of sending to the shader.

The NumGlslVars variable is a sentinel value that is ignored by any functions that accept a glsl_var.

4.1.2.3 typedef enum Camera::ViewType Camera::ViewType

The ViewType enumeration lists the various possibilities for the current viewing mode that can be switched between.

The default is PERSPECTIVE.

4.1.3 Member Enumeration Documentation

4.1.3.1 enum Camera::Direction

The Direction enumeration lists all of the possible directions the camera may travel in.

'BEGIN' and 'END' are special sentinel directions for the purposes of iteration, and are ignored by any functions that accept a Direction.

Definition at line 47 of file Camera.hpp.

4.1.3.2 enum Camera::Uniforms

The glsl_var enumeration lists the various variables the Camera class is capable of sending to the shader.

The NumGlslVars variable is a sentinel value that is ignored by any functions that accept a glsl_var.

Definition at line 73 of file Camera.hpp.

4.1.3.3 enum Camera::ViewType

The ViewType enumeration lists the various possibilities for the current viewing mode that can be switched between.

The default is PERSPECTIVE.

Definition at line 63 of file Camera.hpp.

4.1.4 Constructor & Destructor Documentation

4.1.4.1 Camera::Camera (const std::string & *name*, GLuint *gShader*, float *x* = 0.0, float *y* = 0.0, float *z* = 0.0)

Initialization Constructor; sets the x,y,z coordinates explicitly.

Parameters

name	The name of this Camera/Object.
gShader	A handle to this camera's associated shader object.
x	The initial x coordinate.
y	The initial y coordinate.
z	The initial z coordinate.

Definition at line 45 of file Camera.cpp.

4.1.4.2 Camera::Camera (const std::string & *name*, GLuint *gShader*, vec3 & *in*)

Initialization Constructor, uses a vec3 as its initial coordinates.

Parameters

name	The name of this Camera/Object.
gShader	A handle to this camera's associated shader object.
in	A vec3 representing the initial coordinates.

Definition at line 52 of file Camera.cpp.

4.1.4.3 Camera::Camera (const std::string & *name*, GLuint *gShader*, vec4 & *in*)

Initialization Constructor, uses a vec4 as its initial coordinates.

Parameters

name	The name of this Camera/Object.
gShader	A handle to this camera's associated shader object.
in	A vec4 representing the initial coordinates. The w component is ignored.

Definition at line 58 of file Camera.cpp.

4.1.4.4 Camera::~Camera (void) [virtual]

Default destructor.

Defined only to allow inheritance.

Definition at line 64 of file Camera.cpp.

4.1.5 Member Function Documentation

4.1.5.1 void Camera::accel (const vec3 & *accel*)

accel takes an input vec2 which represents an acceleration, and applies it to the motion vectors with regards to the maximum acceleration and the maximum speed of the camera.

Parameters

accel	The vec3 which represents the (x,y,z) acceleration, where x,y,z are [-1,1].

Returns

Void.

Definition at line 223 of file Camera.cpp.

4.1.5.2 void Camera::adjustFieldOfView (const float & *by*)

adjustFieldOfView adjusts the field of view angle up or down by an amount.

Parameters

by	The float to adjust the fieldOfView angle by.

Returns

Void.

Definition at line 392 of file Camera.cpp.

4.1.5.3 void Camera::adjustRotation (const mat4 & *adjustment,* const bool & *fixed* = `false`) [private]

adjustRotation is an internal function that rotates the camera.

Technically, any transformation, not just a rotation, is possible.

Parameters

adjustment	The 4x4 matrix to transform the CTM by.
fixed	Should this rotation be fixed about the origin?

Returns

Void.

Definition at line 148 of file Camera.cpp.

4.1.5.4 void Camera::changePerspective (const ViewType & *vType*)

changePerspective changes the current perspective of the camera.

Parameters

vType	Which perspective to use. see enum ViewType for possibilities.

Returns

Void.

Definition at line 359 of file Camera.cpp.

4.1.5.5 void Camera::commonInit (void) [private]

commonInit is a private function that initializes local object attributes.

It should be called by all available constructors.

Returns

Void.

Definition at line 19 of file Camera.cpp.

4.1.5.6 void Scene::DeleteObject (Object * *obj*) [protected],[inherited]

DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.

Parameters

obj	The pointer to the object to free.

Definition at line 76 of file Scene.cpp.

4.1.5.7 void Camera::dPos (const float & *x*, const float & *y*, const float & *z*)

Moves the camera along the x, y, and z axes.

Parameters

x	the x-axis displacement.
y	the y-axis displacement.
z	the z-axis displacement.

Returns

Void.

Definition at line 131 of file Camera.cpp.

4.1.5.8 void Camera::dPos (const vec3 & *by*)

Moves the camera along the x, y, and z axes.

Parameters

by	A vec3 containing the x, y, and z axis displacements.

Returns

Void.

Definition at line 140 of file Camera.cpp.

4.1.5.9 void Camera::dPos (const vec4 & *by*)

Moves the camera along the x, y, and z axes.

Parameters

by	A vec4 containing the x, y, and z axis displacements. The w component is ignored.

Returns

Void.

Definition at line 144 of file Camera.cpp.

4.1.5.10 void Camera::dX (const float & *by*, const bool & *update* = `true`)

Moves the camera along the x axis.

Parameters

by	The float value of the x-axis displacement.
update	A boolean indicating whether or not to update the shader. update defaults to true.

Returns

void.

Definition at line 119 of file Camera.cpp.

4.1.5.11 void Camera::dY (const float & *by*, const bool & *update* = `true`)

Moves the camera along the y axis.

Parameters

by	The float value of the y-axis displacement.
update	A boolean indicating whether or not to update the shader. update defaults to true.

Returns

Void.

Definition at line 123 of file Camera.cpp.

4.1.5.12 void Camera::dZ (const float & *by*, const bool & *update* = `true`)

Moves the camera along the z axis.

Parameters

by	The float value of the z-axis displacement.
update	A boolean indicating whether or not to update the shader. update defaults to true.

Returns

Void.

Definition at line 127 of file Camera.cpp.

4.1.5.13 void Camera::fieldOfView (const float & *fovy*)

fieldOfView sets the current camera Field-of-view angle.

This function will send the new perspective matrix to the shader.

Parameters

fovy	The new field of view angle.

Returns

Void.

Definition at line 354 of file Camera.cpp.

4.1.5.14 float Camera::fieldOfView (void) const

fieldOfView() gets the current camera Field-of-view angle.

Returns

A float that is the y axis viewing angle.

Definition at line 350 of file Camera.cpp.

4.1.5.15 vec4 Object::GetPosition ( ) const [inherited]

returns the position of the object this makes the lighting implementation much easier...

for this semester.

Definition at line 497 of file Object.cpp.

4.1.5.16 void Camera::heave (const float & *by*)

Adjusts the camera's y coordinate relative to its current position.

Positive values move the camera up, and negative values move the camera down.

Parameters

by	The float to adjust the y coordinate by.

Returns

Void.

Definition at line 194 of file Camera.cpp.

4.1.5.17 void Camera::idle (void)

idle moves the camera forward in whichever directions it is configured to move in.

Call it in the glut idle function.

Returns

Void.

Definition at line 280 of file Camera.cpp.

4.1.5.18 void Camera::move (const Camera::Direction & *Dir*)

move instructs the camera to begin moving in the specified direction.

Parameters

Dir	The direction in which to move. Can be any direction in the enumerated type Camera::-Direction.

Returns

Void.

Definition at line 272 of file Camera.cpp.

4.1.5.19 void Camera::pitch (const float & *by*, const bool & *fixed* = `false`)

pitch adjusts the x axis rotation; up/down look.

A positive value represents looking up, while a negative value represents looking down.

Parameters

by	A float, in degrees, to adjust the pitch by.
fixed	Should this rotation be fixed about the origin?

Returns

Void.

Definition at line 198 of file Camera.cpp.

4.1.5.20 void Camera::pos (const float & *x*, const float & *y*, const float & *z*, const bool & *update* = `true`)

Sets the absolute position of the camera.

Parameters

x	The new x coordinate of the camera.
y	The new y coordinate of the camera.
z	The new z coordinate of the camera.
update	Whether or not to update the shader with the new coordinates.

Returns

Void.

Definition at line 99 of file Camera.cpp.

4.1.5.21 void Camera::pos (const vec3 & *in*, const bool & *update* = `true`)

Sets the absolute position of the camera.

Parameters

in	A vec3 containing the x, y, and z coordinates to set the camera to.
update	Whether or not to update the shader with the new coordinates.

Returns

Void.

Definition at line 115 of file Camera.cpp.

4.1.5.22 void Camera::pos (const vec4 & *in*, const bool & *update* = `true`)

Sets the absolute position of the camera.

Parameters

in	A vec4 containing the x, y, and z coordinates to set the camera to. The w coordinate is ignored.
update	Whether or not to update the shader with the new coordinates.

Returns

Void.

Definition at line 111 of file Camera.cpp.

4.1.5.23 vec4 Camera::pos (void) const

pos() gets the current camera position in model coordinates.

Returns

A vec4 that represents the current camera coordinates.

Definition at line 346 of file Camera.cpp.

4.1.5.24 void Camera::refreshPerspective (void)

refreshPerspective re-generates the current view/perspective matrix of the camera.

This function should be called after physical or virtual (viewport) screen resizes.

Returns

Void.

Definition at line 366 of file Camera.cpp.

4.1.5.25 void Camera::resetRotation (void)

resetRotation adjusts the camera's rotational state back to its default state (The Identity Matrix.)

Returns

void.

Definition at line 446 of file Camera.cpp.

4.1.5.26 void Camera::roll (const float & *by*, const bool & *fixed* = `false`)

roll adjusts the z axis rotation; tilt or lean left/right.

A positive value represents leaning right, while a negative value represents leaning left.

Parameters

by	A float, in degrees, to adjust the roll by.
fixed	Should this rotation be fixed about the origin?

Returns

Void.

Definition at line 219 of file Camera.cpp.

4.1.5.27 void Camera::send (Object::UniformEnum *which*) `[virtual]`

send will send a glsl variable to the shader.

Parameters

which	The parameter to send. Can be any from enum glsl_var.

Returns

Void.

Reimplemented from Object.

Definition at line 403 of file Camera.cpp.

4.1.5.28 GLuint Object::Shader (void) `[virtual]`, `[inherited]`

Returns the Object's current Shader.

Defined because C++ will not let you overload an overrided function, without re-overloading it in the derived class.

Returns

a GLuint handle to the shader program used by this Object.

Definition at line 269 of file Object.cpp.

4.1.5.29 void Object::Shader (GLuint *newShader*) `[virtual]`, `[inherited]`

Sets the shader to be used by this object.

Triggers a query of the shader program, for the locations of the Uniform locations that the object needs.

Parameters

newShader	a GLuint handle to the shader program to use.

Returns

None.

Reimplemented from Scene.

Definition at line 246 of file Object.cpp.

4.1.5.30 void Camera::stop (const Camera::Direction & *Dir*)

stop instructs the camera to stop moving in the specified direction.

Parameters

Dir	The direction in which to stop moving.

Returns

Void.

Definition at line 276 of file Camera.cpp.

4.1.5.31 void Camera::surge (const float & *by*)

Adjusts the camera's z coordinate relative to its current position.

Positive values move the camera forward, and negative values move the camera backward. Note that the camera uses model coordinates internally, so moving forward will increase the camera's z position negatively.

Parameters

by	The float to adjust the z coordinate by.

Returns

Void.

Definition at line 190 of file Camera.cpp.

4.1.5.32 void Camera::sway (const float & *by*)

Adjusts the camera's x coordinate relative to its current position.

Negative values move the camera left, and positive values move the camera right.

Parameters

by	The float to adjust the x coordinate by.

Returns

Void.

Definition at line 186 of file Camera.cpp.

4.1.5.33 void Camera::view (void)

view will instruct OpenGL of the viewport we want, and then send all of our current matrices to the shader for rendering.

Returns

Void.

Definition at line 434 of file Camera.cpp.

4.1.5.34 void Camera::viewport (size_t *_X*, size_t *_Y*, size_t *_width*, size_t *_height*)

viewport instructs this camera what his expected drawing window will be.

This allows the camera to generate his viewing matrices with the correct aspect ratio.

Parameters

_X	The x coordinate of the lower-left corner of our viewport.
_Y	the y coordinate of the lower-left corner of our viewport.
_width	The width of our viewport.
_height	the height of our viewport.

Returns

Void.

Definition at line 396 of file Camera.cpp.

4.1.5.35 void Camera::x (const float & *in*, const bool & *update* = `true`)

Sets the x coordinate of the camera.

Parameters

in	The new x coordinate of the camera.
update	Whether or not to update the shader with the new coordinates.

Returns

Void.

Definition at line 68 of file Camera.cpp.

4.1.5.36 float Camera::x (void) const

x() returns the current position of the camera in model coordinates.

Returns

The current x coordinate of the camera in model coordinates.

Definition at line 334 of file Camera.cpp.

4.1.5.37 void Camera::y (const float & *in*, const bool & *update* = `true`)

Sets the y coordinate of the camera.

Parameters

in	The new y coordinate of the camera.
update	Whether or not to update the shader with the new coordinates.

Returns

Void.

Definition at line 79 of file Camera.cpp.

4.1.5.38 float Camera::y (void) const

y() returns the current position of the camera in model coordinates.

Returns

The current y coordinate of the camera in model coordinates.

Definition at line 338 of file Camera.cpp.

4.1.5.39 void Camera::yaw (const float & *by*, const bool & *fixed* = `false`)

yaw adjusts the y axis rotation; left/right look.

A positive value represents looking right, while a negative value represents looking left.

Parameters

by	A float, in degrees, to adjust the yaw by.
fixed	Should this rotation be fixed about the origin?

Returns

Void.

Definition at line 209 of file Camera.cpp.

4.1.5.40 void Camera::z (const float & *in*, const bool & *update* = `true`)

Sets the z coordinate of the camera.

Parameters

in	The new z coordinate of the camera.
update	Whether or not to update the shader with the new coordinates.

Returns

Void.

Definition at line 89 of file Camera.cpp.

4.1.5.41 float Camera::z (void) const

z() returns the current position of the camera in model coordinates.

Returns

The current z coordinate of the camera in model coordinates.

Definition at line 342 of file Camera.cpp.

4.1.6 Member Data Documentation

4.1.6.1 GLfloat Camera::_aspectRatio [private]

Current aspect ratio for certain perspectives.

Definition at line 428 of file Camera.hpp.

4.1.6.2 TransCache Camera::_ctm [private]

The Current Transformation state for this Camera.

Definition at line 404 of file Camera.hpp.

4.1.6.3 ViewType Camera::_currentView [private]

The current viewing mode type.

Definition at line 407 of file Camera.hpp.

4.1.6.4 GLfloat Camera::_fovy [private]

Current field-of-view angle for perspective view.

Definition at line 431 of file Camera.hpp.

4.1.6.5 GLfloat Camera::_frictionMagnitude [private]

Friction.

Should be less than MaxAccel.

Definition at line 425 of file Camera.hpp.

4.1.6.6 bool Camera::_motion[Camera::DIR_END] [private]

Booleans correlating to the different motion directions.

Definition at line 440 of file Camera.hpp.

4.1.6.7 GLfloat Camera::_speed [private]

Current Speed of camera motion.

Definition at line 410 of file Camera.hpp.

4.1.6.8 vec3 Camera::_velocity [private]

Current Velocity of camera motion.

Definition at line 413 of file Camera.hpp.

4.1.6.9 mat4 Camera::_view [private]

The current view matrix (defaultly perspective) for this camera.

Definition at line 401 of file Camera.hpp.

4.1.6.10 GLenum Object::draw_mode [protected], [inherited]

Drawing mode for this object.

GL_TRIANGLES, GL_LINE_LOOP, etc.

Definition at line 95 of file Object.hpp.

4.1.6.11 std::vector< GLint > Object::handles [protected], [inherited]

Handles to Uniforms on the shader.

Private to allow derived classes to extend it as needed.

Definition at line 114 of file Object.hpp.

4.1.6.12 std::string Object::name [protected], [inherited]

name is used as an identifying handle for the object.

Definition at line 86 of file Object.hpp.

4.1.6.13 GLuint Object::vao [protected], [inherited]

Vertex Array Object handle identifying our buffers/object.

Definition at line 89 of file Object.hpp.

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

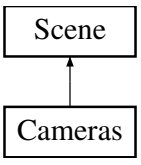
- Camera.hpp
- Camera.cpp

4.2 Cameras Class Reference

The Cameras class represents a group of logical cameras for a model view. Each camera possesses its own current viewing angle, and an absolute position in space.

`#include <Cameras.hpp>`

Inheritance diagram for Cameras:



Public Member Functions

- Cameras (void)

  Default constructor.
- ~Cameras (void)

  Default destructor.
- Camera * addCamera (const std::string &name)

  addCamera takes a name for a camera and returns a handle to a newly created camera.

- void popCamera (void)

  popCamera removes the most recently added Camera from the scene.
- Camera * next (void)

  Sets the active camera to the next available one in the collection.
- Camera * prev (void)

  Sets the active Camera to the previous available one in the collection.
- Camera * active (void) const

  active returns the Camera in the collection that is considered 'active'.
- size_t numCameras (void) const

  numCameras fetches the number of Cameras in the collection.
- void idleMotion (void)

  idleMotion calls the idle method on all child cameras.
- void resize (int width, int height)

  resize informs the Cameras collection of the new window size.
- void calculateViewports (void)

  For each Camera in the collection, computes the position and size of that Camera's viewport in a split-screen, single-window configuration.
- void view (void(*draw_func)(void))

  view calls the view method on all child cameras, followed by the provided draw function.
- Camera * obj2Cam (std::list< Object * >::iterator &it)

  obj2Cam is a gross hack; the function is used as a utility to convert Object pointers to Camera pointers safely.
- virtual void Shader (GLuint gShader)

  Sets the Default shader for the scene.
- GLuint Shader (void)

  Retrieves the handle for the default shader for the scene.
- Object * **AddObject** (const std::string &objName, GLuint Object_Shader=0)
- void **DelObject** (const std::string &objName)
- void **DelObject** (void)
- void **PopObject** (void)
- void DestroyObject (void)

  Completely remove this object and all his children.
- void **Draw** (void)
- Object * **operator[]** (const std::string &objname)

Protected Member Functions

- void DeleteObject (Object *obj)

  DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.
- void **InsertObject** (const std::string name, Object *obj)

Protected Attributes

- std::list< Object * > **_list**
- std::map< std::string, Object * > **_map**
- std::list< Object * >::iterator **_currentObj**
- GLuint **_gShader**

Private Attributes

- Angel::vec2 _size

  _size is a simple vec2 (x,y) that contains the size of the screen.

4.2.1 Detailed Description

The Cameras class represents a group of logical cameras for a model view. Each camera possesses its own current viewing angle, and an absolute position in space.

Author

John Huston, jhuston@cs.uml.edu

Since

28 Nov 2012

Each Camera possesses its own CTM which can be resent to the GPU at will.

Definition at line 29 of file Cameras.hpp.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Cameras::Cameras (void)

Default constructor.

Nothing special.

Definition at line 19 of file Cameras.cpp.

4.2.2.2 Cameras::~Cameras (void)

Default destructor.

Nothing special here, either.

Definition at line 26 of file Cameras.cpp.

4.2.3 Member Function Documentation

4.2.3.1 Camera * Cameras::active (void) const

active returns the Camera in the collection that is considered 'active'.

Returns

A pointer to the currently selected, active Camera.

Definition at line 86 of file Cameras.cpp.

4.2.3.2 Camera * Cameras::addCamera (const std::string & *name*)

addCamera takes a name for a camera and returns a handle to a newly created camera.

Parameters

name	The name of the new camera to create.

Returns

A Pointer to a newly created Camera object.

Definition at line 35 of file Cameras.cpp.

4.2.3.3 void Cameras::calculateViewports (void)

For each Camera in the collection, computes the position and size of that Camera's viewport in a split-screen, single-window configuration.

The Camera object is updated with the new information.

Returns

void.
void.

Definition at line 141 of file Cameras.cpp.

4.2.3.4 void Scene::DeleteObject (Object * *obj*) [protected], [inherited]

DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.

Parameters

obj	The pointer to the object to free.

Definition at line 76 of file Scene.cpp.

4.2.3.5 void Cameras::idleMotion (void)

idleMotion calls the idle method on all child cameras.

Intended to be called during the idle() loop in GLUT.

Returns

void.

Definition at line 111 of file Cameras.cpp.

4.2.3.6 Camera * Cameras::next (void)

Sets the active camera to the next available one in the collection.

Sets the active Camera to the next available one in the collection.

Returns

A pointer to the newly active Camera.

Definition at line 64 of file Cameras.cpp.

4.2.3.7 size_t Cameras::numCameras (void) const

numCameras fetches the number of Cameras in the collection.

Returns

an unsigned integer, the number of Cameras in the collection.

Definition at line 101 of file Cameras.cpp.

4.2.3.8 Camera * Cameras::obj2Cam (std::list< Object * >::iterator & *it*)

obj2Cam is a gross hack; the function is used as a utility to convert Object pointers to Camera pointers safely.

FIXME: Refactor the inheritance here to make this less hacky.

Parameters

it	A list<Object*> iterator that points to the Object.

Returns

A pointer to a Camera object.

Definition at line 249 of file Cameras.cpp.

4.2.3.9 void Cameras::popCamera (void)

popCamera removes the most recently added Camera from the scene.

Returns

void.

Definition at line 50 of file Cameras.cpp.

4.2.3.10 Camera * Cameras::prev (void)

Sets the active Camera to the previous available one in the collection.

Returns

A pointer to the newly active Camera.

Definition at line 75 of file Cameras.cpp.

4.2.3.11 void Cameras::resize (int *width*, int *height*)

resize informs the Cameras collection of the new window size.

Intended to be called from the GLUT main loop. This method also invokes Cameras::calculateViewports.

Parameters

width	The new window width.
height	The new window height.

Returns

void.

Definition at line 128 of file Cameras.cpp.

4.2.3.12 void Scene::Shader (GLuint *gShader*) [virtual],[inherited]

Sets the Default shader for the scene.

In the context of inheritance by objects, This sets the shader to use to render the physical object.

Parameters

gShader	The GLuint handle to the shader to use.

Returns

void.

Reimplemented in Object.

Definition at line 54 of file Scene.cpp.

4.2.3.13 GLuint Scene::Shader (void) [inherited]

Retrieves the handle for the default shader for the scene.

In the context of inheritance by objects, This retrieves the shader handle to use to draw the object.

Returns

A GLuint handle to the shader program.

Definition at line 66 of file Scene.cpp.

4.2.3.14 void Cameras::view (void(*)(void) *draw_func*)

view calls the view method on all child cameras, followed by the provided draw function.

Intended to be called during the display() portion of the GLUT main loop.

view() is intended to "set up" the object, but not actually draw it.

Parameters

draw_func	A pointer to a function that will actually draw the object.

Definition at line 232 of file Cameras.cpp.

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

- Cameras.hpp
- Cameras.cpp

4.3 Engine Class Reference

The Engine class is a singleton-style class which helps keep track of instances of important objects (for Cameras, Objects, etc) as well as some settings and variables that would otherwise clog up global namespace.

```
#include <Engine.hpp>
```

Public Member Functions

- Cameras * cams (void)

Retrieves a pointer to the Camera List.

- Scene * rootScene (void)

  Retrieves a pointer to the Root, top-level Scene graph.

- Screen * mainScreen (void)

  Retrieves a pointer to the core 'Screen' object.

- bool opt (const std::string &Option)

  opt retrieves the current setting of an option in the Engine.

- void opt (const std::string &Option, bool setting)

  opt, with a second parameter, sets an Engine option.

- bool set (const std::string &Option)

  set checks to see if an option has been explicitly set to either True/False.

- bool flip (const std::string &Option)

  flip changes a setting from its current value to its negated form.

- ~Engine (void)

  Default, non-virtual destructor.

Static Public Member Functions

- static Engine * instance (void)

  instance returns, or creates and then returns, a pointer to the Engine object.

Private Member Functions

- Engine (void)

  Default constructor.

- Engine (const Engine &copy)

  Copy constructor.

- Engine & operator= (Engine &assign)

  Assignment operator.

Private Attributes

- Scene _scene

  The root Scene graph for the Engine.

- Screen _screen

  The core "Screen" object for the Engine.

- SettingsMap _engineSettings

  _engineSettings is a std::map that contains a series of <std::string, bool> pairs that represent our Engine options.

Static Private Attributes

- static Engine * _engineSingleton = NULL

  static, stateful variable that is our singleton pointer.

4.3.1 Detailed Description

The Engine class is a singleton-style class which helps keep track of instances of important objects (for Cameras, Objects, etc) as well as some settings and variables that would otherwise clog up global namespace.

Author

John Huston, jhuston@cs.uml.edu

Date

2013-03-13

Definition at line 32 of file Engine.hpp.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Engine::Engine (void) [private]

Default constructor.

Cannot be called, this is a singleton class.

Definition at line 40 of file Engine.cpp.

4.3.2.2 Engine::Engine (const Engine & *copy*) [private]

Copy constructor.

Cannot be called.

Parameters

copy	Not used.

Returns

Will always throw an exception.

Exceptions

Will	always throw std::logic_error.

Definition at line 57 of file Engine.cpp.

4.3.3 Member Function Documentation

4.3.3.1 Cameras * Engine::cams (void)

Retrieves a pointer to the Camera List.

Returns

A pointer to the Camera List.

Definition at line 76 of file Engine.cpp.

4.3.3.2 bool Engine::flip (const std::string & *Option*)

flip changes a setting from its current value to its negated form.

Parameters

Option	The option to toggle.

Returns

The new, current value of the option.

Definition at line 139 of file Engine.cpp.

4.3.3.3 Engine * Engine::instance (void) [static]

instance returns, or creates and then returns, a pointer to the Engine object.

All hail the singleton!

Returns

A pointer to the Engine object.

Definition at line 29 of file Engine.cpp.

4.3.3.4 Screen * Engine::mainScreen (void)

Retrieves a pointer to the core 'Screen' object.

Returns

A pointer to the core 'Screen' object.

Definition at line 96 of file Engine.cpp.

4.3.3.5 Engine & Engine::operator= (Engine & *assign*) [private]

Assignment operator.

Cannot be used. This is a singleton class.

Parameters

assign	Not used.

Returns

Will always throw an exception.

Exceptions

Will	always throw std::logic_error.

Definition at line 67 of file Engine.cpp.

4.3.3.6 bool Engine::opt (const std::string & *Option*)

opt retrieves the current setting of an option in the Engine.

Parameters

Option	The name of the option to access.

Returns

A bool: the current value of the setting.

Definition at line 107 of file Engine.cpp.

4.3.3.7 void Engine::opt (const std::string & *Option,* bool *setting*)

opt, with a second parameter, sets an Engine option.

Parameters

Option	The name of the option to set.
setting	The value to give the option.

Returns

void.

Definition at line 117 of file Engine.cpp.

4.3.3.8 Scene * Engine::rootScene (void)

Retrieves a pointer to the Root, top-level Scene graph.

Returns

A pointer to the Root-level Scene graph.

Definition at line 86 of file Engine.cpp.

4.3.3.9 bool Engine::set (const std::string & *Option*)

set checks to see if an option has been explicitly set to either True/False.

Parameters

Option	The option to check the existence of

Returns

A boolean: True if the option has been set, False otherwise.

Definition at line 127 of file Engine.cpp.

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

- Engine.hpp
- Engine.cpp

4.4 TiemSpelchk::Lurn2SpielNub Class Reference

Public Member Functions

- void **setCallback** (boost::function< void(int, double, double, double)> headCB)
- int **Start** ()
- void **Shutdown** ()

Static Public Member Functions

- static void XN_CALLBACK_TYPE **new_user** (xn::UserGenerator &, XnUserID, void *)
- static void XN_CALLBACK_TYPE **lost_user** (xn::UserGenerator &, XnUserID, void *)
- static void XN_CALLBACK_TYPE **pose** (xn::PoseDetectionCapability &, const XnChar *, XnUserID, void *)
- static void XN_CALLBACK_TYPE **cal_start** (xn::SkeletonCapability &, XnUserID, void *)
- static void XN_CALLBACK_TYPE **cal_complete** (xn::SkeletonCapability &, XnUserID, XnCalibrationStatus, void *)

Private Member Functions

- void **FUNKMASTER_thread_func** ()
- void XN_CALLBACK_TYPE **User_NewUser** (xn::UserGenerator &, XnUserID nId, void *)
- void XN_CALLBACK_TYPE **User_LostUser** (xn::UserGenerator &, XnUserID nId, void *)
- void XN_CALLBACK_TYPE **UserPose_PoseDetected** (xn::PoseDetectionCapability &, const XnChar *strPose, XnUserID nId, void *)
- void XN_CALLBACK_TYPE **UserCalibration_CalibrationStart** (xn::SkeletonCapability &, XnUserID nId, void *)
- void XN_CALLBACK_TYPE **UserCalibration_CalibrationComplete** (xn::SkeletonCapability &, XnUserID nId, XnCalibrationStatus eStatus, void *)

Private Attributes

- boost::function< void(int, double, double, double)> **_cb**
- boost::thread **_thread**
- bool **needsToSeppuku**
- xn::Context **g_Context**
- xn::ScriptNode **g_scriptNode**
- xn::DepthGenerator **g_DepthGenerator**
- xn::UserGenerator **g_UserGenerator**
- XnBool **g_bNeedPose**
- XnChar **g_strPose** [20]

4.4.1 Detailed Description

Definition at line 58 of file KinectInator.hpp.

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

- KinectInator.hpp
- KinectInator.cpp

4.5 Angel::mat2 Class Reference

mat2 - 2D square matrix.

`#include <mat.hpp>`

Public Member Functions

- **mat2** (const GLfloat d=GLfloat(1.0))
- **mat2** (const vec2 &a, const vec2 &b)
- **mat2** (GLfloat m00, GLfloat m10, GLfloat m01, GLfloat m11)
- **mat2** (const mat2 &m)
- vec2 & **operator[]** (int i)
- const vec2 & **operator[]** (int i) const
- mat2 **operator+** (const mat2 &m) const
- mat2 **operator-** (const mat2 &m) const
- mat2 **operator*** (const GLfloat s) const
- mat2 **operator/** (const GLfloat s) const
- mat2 **operator*** (const mat2 &m) const
- mat2 & **operator+=** (const mat2 &m)
- mat2 & **operator-=** (const mat2 &m)
- mat2 & **operator*=** (const GLfloat s)
- mat2 & **operator*=** (const mat2 &m)
- mat2 & **operator/=** (const GLfloat s)
- vec2 operator* (const vec2 &v) const

  *Returns the result of the operation M*v.*
- **operator const GLfloat** * () const
- **operator GLfloat** * ()

Private Attributes

- vec2 _m [2]

  The data is stored as an array of two vectors.

Friends

- mat2 **operator*** (const GLfloat s, const mat2 &m)
- std::ostream & **operator<<** (std::ostream &os, const mat2 &m)
- std::istream & **operator>>** (std::istream &is, mat2 &m)

4.5.1 Detailed Description

mat2 - 2D square matrix.

Author

Ed Angel Class for a 2x2 matrix. Modified from code available from Ed Angel's website, http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/ Published from his book, Interactive Computer Graphics A Top-Down Approach with OpenGL, Sixth Edition

Definition at line 36 of file mat.hpp.

4.5.2 Member Function Documentation

4.5.2.1 vec2 Angel::mat2::operator∗ (const vec2 & *v*) const

Returns the result of the operation M∗v.

Assumes v is a one column, two row matrix.

Definition at line 154 of file mat.cpp.

4.5.3 Member Data Documentation

4.5.3.1 vec2 Angel::mat2::_m[2] [private]

The data is stored as an array of two vectors.

Definition at line 39 of file mat.hpp.

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

- mat.hpp
- mat.cpp

4.6 Angel::mat3 Class Reference

mat3 - 3D square matrix.

```
#include <mat.hpp>
```

Public Member Functions

- **mat3** (const GLfloat d=GLfloat(1.0))
- **mat3** (const vec3 &a, const vec3 &b, const vec3 &c)
- **mat3** (GLfloat m00, GLfloat m10, GLfloat m20, GLfloat m01, GLfloat m11, GLfloat m21, GLfloat m02, GLfloat m12, GLfloat m22)
- **mat3** (const mat3 &m)
- vec3 & **operator[]** (int i)
- const vec3 & **operator[]** (int i) const
- mat3 **operator+** (const mat3 &m) const
- mat3 **operator-** (const mat3 &m) const
- mat3 **operator∗** (const GLfloat s) const
- mat3 **operator/** (const GLfloat s) const
- mat3 **operator∗** (const mat3 &m) const
- mat3 & **operator+=** (const mat3 &m)
- mat3 & **operator-=** (const mat3 &m)
- mat3 & **operator∗=** (const GLfloat s)
- mat3 & **operator∗=** (const mat3 &m)
- mat3 & **operator/=** (const GLfloat s)
- vec3 **operator∗** (const vec3 &v) const
- **operator const GLfloat** ∗ () const
- **operator GLfloat** ∗ ()

Private Attributes

- vec3 **_m** [3]

Friends

- mat3 **operator**∗ (const GLfloat s, const mat3 &m)
- std::ostream & **operator**<< (std::ostream &os, const mat3 &m)
- std::istream & **operator**>> (std::istream &is, mat3 &m)

4.6.1 Detailed Description

mat3 - 3D square matrix.

Author

Ed Angel Class for a 2x2 matrix. Modified from code available from Ed Angel's website, http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/ Published from his book, Interactive Computer Graphics A Top-Down Approach with OpenGL, Sixth Edition

Definition at line 94 of file mat.hpp.

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

- mat.hpp
- mat.cpp

4.7 Angel::mat4 Class Reference

mat4 - 4D square matrix.

`#include <mat.hpp>`

Public Member Functions

- **mat4** (const GLfloat d=GLfloat(1.0))
- **mat4** (const vec4 &a, const vec4 &b, const vec4 &c, const vec4 &d)
- **mat4** (GLfloat m00, GLfloat m01, GLfloat m02, GLfloat m03, GLfloat m10, GLfloat m11, GLfloat m12, GLfloat m13, GLfloat m20, GLfloat m21, GLfloat m22, GLfloat m23, GLfloat m30, GLfloat m31, GLfloat m32, GLfloat m33)
- **mat4** (const mat4 &m)
- vec4 & **operator[]** (int i)
- const vec4 & **operator[]** (int i) const
- mat4 **operator+** (const mat4 &m) const
- mat4 **operator-** (const mat4 &m) const
- mat4 **operator**∗ (const GLfloat s) const
- mat4 **operator**/ (const GLfloat s) const
- mat4 **operator**∗ (const mat4 &m) const
- mat4 & **operator+=** (const mat4 &m)
- mat4 & **operator-=** (const mat4 &m)
- mat4 & **operator**∗**=** (const GLfloat s)
- mat4 & **operator**∗**=** (const mat4 &m)
- mat4 & **operator/=** (const GLfloat s)
- vec4 **operator**∗ (const vec4 &v) const
- **operator const GLfloat** ∗ () const
- **operator GLfloat** ∗ ()

Private Attributes

- vec4 **_m** [4]

Friends

- mat4 **operator**∗ (const GLfloat s, const mat4 &m)
- vec4 **operator**∗ (const vec4 &v, const mat4 &_m)
- std::ostream & **operator**<< (std::ostream &os, const mat4 &m)
- std::istream & **operator**>> (std::istream &is, mat4 &m)

4.7.1 Detailed Description

mat4 - 4D square matrix.

Author

Ed Angel Class for a 2x2 matrix. Modified from code available from Ed Angel's website, http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/ Published from his book, Interactive Computer Graphics A Top-Down Approach with OpenGL, Sixth Edition

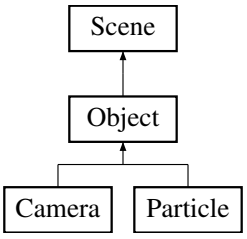
Definition at line 150 of file mat.hpp.

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

- mat.hpp
- mat.cpp

4.8 Object Class Reference

Inheritance diagram for Object:



Public Types

- enum **Uniforms** {
  BEGIN, **IsTextured** = BEGIN, **ObjectCTM**, **MorphPercentage**,
  END }
- typedef const unsigned int **UniformEnum**
- typedef std::map
  < Object::UniformEnum,
  std::string > **UniformMap**
- typedef enum Object::Uniforms **Uniform**

Public Member Functions

- **Object** (const std::string &name, GLuint gShader)
- void **Draw** (void)
- void **Buffer** (void)
- void **BufferMorphOnly** (void)
- void **Mode** (GLenum new_node)
- void **Texture** (const char **filename)
- const std::string & **Name** (void) const
- virtual void **Link** (UniformEnum which, const std::string &name)
- virtual void **send** (UniformEnum which)
- virtual GLuint Shader (void)

  Returns the Object's current Shader.
- virtual void Shader (GLuint newShader)

  Sets the shader to be used by this object.
- void **Animation** (void(*anim_func)(TransCache &arg))
- void **Propagate** (void)
- vec4 GetPosition () const

  returns the position of the object this makes the lighting implementation much easier...
- Object * **getMorphTargetPtr** () const
- Object * **genMorphTarget** (GLuint)
- float **getMorphPercentage** () const
- void **setMorphPercentage** (const float)
- void **destroyMorphTarget** ()
- int **getNumberPoints** ()
- Object * **AddObject** (const std::string &objName, GLuint Object_Shader=0)
- void **DelObject** (const std::string &objName)
- void **DelObject** (void)
- void **PopObject** (void)
- void DestroyObject (void)

  Completely remove this object and all his children.
- Object * **next** (void)
- Object * **prev** (void)
- Object * **active** (void) const
- Object * **operator[]** (const std::string &objname)

Public Attributes

- std::vector< Angel::vec4 > **points**
- std::vector< Angel::vec3 > **normals**
- std::vector< unsigned int > **indices**
- std::vector< Angel::vec4 > **colors**
- std::vector< Angel::vec2 > **texcoords**
- TransCache **trans**

Protected Member Functions

- void DeleteObject (Object *obj)

  DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.
- void **InsertObject** (const std::string name, Object *obj)

Protected Attributes

- std::string name

  name is used as an identifying handle for the object.
- GLuint vao

  Vertex Array Object handle identifying our buffers/object.
- GLuint buffer [NUM_BUFFERS]

  Handles to our buffers (Vertices, TexUVs, etc.)
- GLenum draw_mode

  Drawing mode for this object.
- bool isTextured

  Is this object textured?
- float morphPercentage

  Morphing/Tweening Things.
- Object * **morphTarget**
- std::map< Object::UniformEnum, std::string > **_uniformMap**
- std::vector< GLint > handles

  Handles to Uniforms on the shader.
- std::list< Object * > **_list**
- std::map< std::string, Object * > **_map**
- std::list< Object * >::iterator **_currentObj**
- GLuint **_gShader**

Private Types

- enum **bufferType** {
  VERTICES, **NORMALS**, **INDICES**, **COLORS**,
  TEXCOORDS, **VERTICES_MORPH**, **NORMALS_MORPH**, **COLORS_MORPH**,
  NUM_BUFFERS }

4.8.1 Detailed Description

Definition at line 16 of file Object.hpp.

4.8.2 Member Function Documentation

4.8.2.1 void Scene::DeleteObject (Object * *obj*) [protected], [inherited]

DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.

Parameters

obj	The pointer to the object to free.

Definition at line 76 of file Scene.cpp.

4.8.2.2 vec4 Object::GetPosition ( ) const

returns the position of the object this makes the lighting implementation much easier...

for this semester.

Definition at line 497 of file Object.cpp.

4.8.2.3 GLuint Object::Shader (void) [virtual]

Returns the Object's current Shader.

Defined because C++ will not let you overload an overrided function, without re-overloading it in the derived class.

Returns

a GLuint handle to the shader program used by this Object.

Definition at line 269 of file Object.cpp.

4.8.2.4 void Object::Shader (GLuint *newShader*) [virtual]

Sets the shader to be used by this object.

Triggers a query of the shader program, for the locations of the Uniform locations that the object needs.

Parameters

newShader	a GLuint handle to the shader program to use.

Returns

None.

Reimplemented from Scene.

Definition at line 246 of file Object.cpp.

4.8.3 Member Data Documentation

4.8.3.1 GLenum Object::draw_mode [protected]

Drawing mode for this object.

GL_TRIANGLES, GL_LINE_LOOP, etc.

Definition at line 95 of file Object.hpp.

4.8.3.2 std::vector< GLint > Object::handles [protected]

Handles to Uniforms on the shader.

Private to allow derived classes to extend it as needed.

Definition at line 114 of file Object.hpp.

4.8.3.3 std::string Object::name [protected]

name is used as an identifying handle for the object.

Definition at line 86 of file Object.hpp.

4.8.3.4 GLuint Object::vao [protected]

Vertex Array Object handle identifying our buffers/object.

Definition at line 89 of file Object.hpp.

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

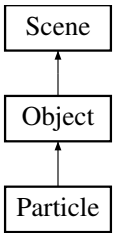
- Object.hpp
- Object.cpp

4.9 Particle Class Reference

todo

`#include <Particle.hpp>`

Inheritance diagram for Particle:



Public Types

- enum **Uniforms** {
  BEGIN, **IsTextured** = BEGIN, **ObjectCTM**, **MorphPercentage**,
  END }
- typedef const unsigned int **UniformEnum**
- typedef std::map
  < Object::UniformEnum,
  std::string > **UniformMap**
- typedef enum Object::Uniforms **Uniform**

Public Member Functions

- **Particle** (vec4 initPos, vec3 initScale, vec3 initVel, float initAlpha, vec4 initColor, float initLifespan, float initSpin, string initTex)
- void **update** ()
- void **setPos** (vec4 newPos)
- void **setScale** (vec3 newScale)
- void **setVel** (vec3 newVel)
- void **setAlpha** (float newAlpha)
- void **setColor** (vec4 newColor)
- void **setLifespan** (float newLifespan)
- void **setSpin** (float newSpin)
- void **setTexFile** (string newFilename)
- void **Draw** (void)
- void **Buffer** (void)
- void **BufferMorphOnly** (void)
- void **Mode** (GLenum new_node)
- void **Texture** (const char **filename)
- const std::string & **Name** (void) const
- virtual void **Link** (UniformEnum which, const std::string &name)

- virtual void **send** (UniformEnum which)
- virtual GLuint Shader (void)

  Returns the Object's current Shader.
- virtual void Shader (GLuint newShader)

  Sets the shader to be used by this object.
- void **Animation** (void(∗anim_func)(TransCache &arg))
- void **Propagate** (void)
- vec4 GetPosition () const

  returns the position of the object this makes the lighting implementation much easier...
- Object ∗ **getMorphTargetPtr** () const
- Object ∗ **genMorphTarget** (GLuint)
- float **getMorphPercentage** () const
- void **setMorphPercentage** (const float)
- void **destroyMorphTarget** ()
- int **getNumberPoints** ()
- Object ∗ **AddObject** (const std::string &objName, GLuint Object_Shader=0)
- void **DelObject** (const std::string &objName)
- void **DelObject** (void)
- void **PopObject** (void)
- void DestroyObject (void)

  Completely remove this object and all his children.
- Object ∗ **next** (void)
- Object ∗ **prev** (void)
- Object ∗ **active** (void) const
- Object ∗ **operator[]** (const std::string &objname)

Public Attributes

- std::vector< Angel::vec4 > **points**
- std::vector< Angel::vec3 > **normals**
- std::vector< unsigned int > **indices**
- std::vector< Angel::vec4 > **colors**
- std::vector< Angel::vec2 > **texcoords**
- TransCache **trans**

Protected Member Functions

- void DeleteObject (Object ∗obj)

  DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.
- void **InsertObject** (const std::string name, Object ∗obj)

Protected Attributes

- std::string name

  name is used as an identifying handle for the object.
- GLuint vao

  Vertex Array Object handle identifying our buffers/object.
- GLuint buffer [NUM_BUFFERS]

  Handles to our buffers (Vertices, TexUVs, etc.)
- GLenum draw_mode

  Drawing mode for this object.

- bool isTextured

  Is this object textured?
- float morphPercentage

  Morphing/Tweening Things.
- Object * **morphTarget**
- std::map< Object::UniformEnum, std::string > **_uniformMap**
- std::vector< GLint > handles

  Handles to Uniforms on the shader.
- std::list< Object * > **_list**
- std::map< std::string, Object * > **_map**
- std::list< Object * >::iterator **_currentObj**
- GLuint **_gShader**

Private Attributes

- vec4 **mPos**
- vec3 **mScale**
- vec3 **mVel**
- float **alpha**
- vec4 **blendColor**
- float **lifespan**
- float **spin**
- string **texFilename**

4.9.1 Detailed Description

todo

Author

Nick Ver Voort, nicholas_vervoort@student.uml.edu

Since

23 Feb 2013

Definition at line 22 of file Particle.hpp.

4.9.2 Member Function Documentation

4.9.2.1 void Scene::DeleteObject (Object * *obj*) [protected], [inherited]

DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.

Parameters

obj	The pointer to the object to free.

Definition at line 76 of file Scene.cpp.

4.9.2.2 vec4 Object::GetPosition ( ) const [inherited]

returns the position of the object this makes the lighting implementation much easier...

for this semester.

Definition at line 497 of file Object.cpp.

4.9.2.3 GLuint Object::Shader (void) [virtual], [inherited]

Returns the Object's current Shader.

Defined because C++ will not let you overload an overrided function, without re-overloading it in the derived class.

Returns

a GLuint handle to the shader program used by this Object.

Definition at line 269 of file Object.cpp.

4.9.2.4 void Object::Shader (GLuint *newShader*) [virtual], [inherited]

Sets the shader to be used by this object.

Triggers a query of the shader program, for the locations of the Uniform locations that the object needs.

Parameters

newShader	a GLuint handle to the shader program to use.

Returns

None.

Reimplemented from Scene.

Definition at line 246 of file Object.cpp.

4.9.3 Member Data Documentation

4.9.3.1 GLenum Object::draw_mode [protected], [inherited]

Drawing mode for this object.

GL_TRIANGLES, GL_LINE_LOOP, etc.

Definition at line 95 of file Object.hpp.

4.9.3.2 std::vector< GLint > Object::handles [protected], [inherited]

Handles to Uniforms on the shader.

Private to allow derived classes to extend it as needed.

Definition at line 114 of file Object.hpp.

4.9.3.3 std::string Object::name [protected], [inherited]

name is used as an identifying handle for the object.

Definition at line 86 of file Object.hpp.

4.9.3.4 GLuint Object::vao [protected], [inherited]

Vertex Array Object handle identifying our buffers/object.

Definition at line 89 of file Object.hpp.

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

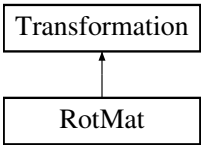
- Particle.hpp
- Particle.cpp

4.10 RotMat Class Reference

Rotations.

`#include <Transformation.hpp>`

Inheritance diagram for RotMat:



Public Member Functions

- const RotMat & **Reset** (const Angel::mat4 &NewState)
- const RotMat & **RotateX** (const GLfloat theta, bool postmult=true)
- const RotMat & **RotateY** (const GLfloat theta, bool postmult=true)
- const RotMat & **RotateZ** (const GLfloat theta, bool postmult=true)
- const RotMat & **Adjust** (const Angel::mat4 &Adjustment, bool postmult=true)
- const Angel::mat4 & **Matrix** (void) const
- Angel::mat4 **operator*** (const Angel::mat4 &rhs) const
- Angel::mat4 **operator*** (const Transformation &rhs) const

Protected Attributes

- Angel::mat4 **mat**

4.10.1 Detailed Description

Rotations.

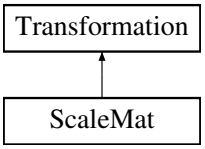
Definition at line 33 of file Transformation.hpp.

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

- Transformation.hpp
- Transformation.cpp

4.11 ScaleMat Class Reference

Inheritance diagram for ScaleMat:



Public Member Functions

- const ScaleMat & **Set** (const float x, const float y, const float z)
- const ScaleMat & **Set** (const float pct)
- const ScaleMat & **Adjust** (const float x, const float y, const float z)
- const ScaleMat & **Adjust** (const float pct)
- const Angel::mat4 & **Matrix** (void) const
- Angel::mat4 **operator**∗ (const Angel::mat4 &rhs) const
- Angel::mat4 **operator**∗ (const Transformation &rhs) const

Protected Attributes

- Angel::mat4 **mat**

4.11.1 Detailed Description

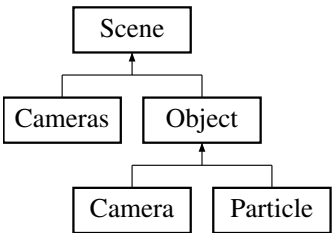
Definition at line 63 of file Transformation.hpp.

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

- Transformation.hpp
- Transformation.cpp

4.12 Scene Class Reference

Inheritance diagram for Scene:



Public Member Functions

- virtual void Shader (GLuint gShader)

  Sets the Default shader for the scene.
- GLuint Shader (void)

Retrieves the handle for the default shader for the scene.

- Object * **AddObject** (const std::string &objName, GLuint Object_Shader=0)
- void **DelObject** (const std::string &objName)
- void **DelObject** (void)
- void **PopObject** (void)
- void DestroyObject (void)

  Completely remove this object and all his children.
- Object * **next** (void)
- Object * **prev** (void)
- Object * **active** (void) const
- void **Draw** (void)
- Object * **operator[]** (const std::string &objname)
- **Scene** (const Scene &copy)
- Scene & **operator=** (const Scene &copy)

Protected Member Functions

- void DeleteObject (Object *obj)

  DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.
- void **InsertObject** (const std::string name, Object *obj)

Protected Attributes

- std::list< Object * > **_list**
- std::map< std::string, Object * > **_map**
- std::list< Object * >::iterator **_currentObj**
- GLuint **_gShader**

4.12.1 Detailed Description

Definition at line 12 of file Scene.hpp.

4.12.2 Member Function Documentation

4.12.2.1 void Scene::DeleteObject (Object * *obj*) [protected]

DeleteObject is the actual implementation function that will remove an Object from the Scene list and Scene map, then free the object.

Parameters

obj	The pointer to the object to free.

Definition at line 76 of file Scene.cpp.

4.12.2.2 void Scene::Shader (GLuint *gShader*) [virtual]

Sets the Default shader for the scene.

In the context of inheritance by objects, This sets the shader to use to render the physical object.

Parameters

gShader	The GLuint handle to the shader to use.

Returns

void.

Reimplemented in Object.

Definition at line 54 of file Scene.cpp.

4.12.2.3 GLuint Scene::Shader (void)

Retrieves the handle for the default shader for the scene.

In the context of inheritance by objects, This retrieves the shader handle to use to draw the object.

Returns

A GLuint handle to the shader program.

Definition at line 66 of file Scene.cpp.

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

- Scene.hpp
- Scene.cpp

4.13 Screen Class Reference

Public Member Functions

- **Screen** (int x=0, int y=0)
- **Screen** (const vec2 &newSize)
- void **Size** (int x, int y)
- void **Size** (const vec2 &newSize)
- const vec2 & **Size** (void)
- int **Width** (void)
- int **Height** (void)
- const vec2 & **Center** (void)
- int **MidpointX** (void)
- int **MidpointY** (void)

Public Attributes

- Cameras **_camList**

Private Attributes

- vec2 **_size**
- vec2 **_center**

4.13.1 Detailed Description

Definition at line 8 of file Screen.hpp.

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

- Screen.hpp
- Screen.cpp

4.14 SpelchkCamera Class Reference

Public Member Functions

- **SpelchkCamera** (vec4 initialTranslationVector)
- mat4 **getProjectionMatrix** ()
- mat4 **getModelViewMatrix** ()
- vec4 **getTranslationVector** ()
- void **moveCamera** (float xDepth, float yDepth, float zDepth)
- void **rotateCamera** (float xAngle, float yAngle, float zAngle)
- void **setScreenSize** (int width, int height)
- void **setProjection** (int projectionType)
- void **reset** ()
- void **setLightMovementRef** (GLuint ref)
- void **setLightMovementTime** (float elapsed)
- void **getReadyForZero** (int usernum)
- void **headMovement** (int usernum, double x, double y, double z)

Private Member Functions

- void **calculateTranslationVector** ()

Private Attributes

- int **projectionType**
- GLfloat **fovy**
- GLfloat **aspect**
- GLfloat **left**
- GLfloat **right**
- GLfloat **bottom**
- GLfloat **top**
- GLfloat **zNear**
- GLfloat **zFar**
- GLuint **timeRef**
- int **screenWidth**
- int **screenHeight**
- GLfloat **xDepth**
- GLfloat **yDepth**
- GLfloat **zDepth**
- GLfloat **xAngle**
- GLfloat **yAngle**
- GLfloat **zAngle**
- GLfloat **xHead**
- GLfloat **yHead**

- GLfloat **zHead**
- float **xHeadStart**
- float **yHeadStart**
- float **zHeadStart**
- GLfloat **xHeadAngle**
- GLfloat **yHeadAngle**
- GLfloat **zHeadAngle**
- vec4 **initialTranslationVector**
- vec4 **translationVector**
- vec4 **oldTranslationVector**
- mat4 **modelViewMatrix**
- int **inboundHeadData**
- vec4 **initialHeadPosition**

4.14.1 Detailed Description

Definition at line 16 of file SpelchkCamera.hpp.

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

- SpelchkCamera.hpp
- SpelchkCamera.cpp

4.15 Timer Class Reference

Public Member Functions

- unsigned long Tick ()

  Tick is an alias for Tock.
- unsigned long Tock ()

  Tock returns the time elapsed since the last Tock.
- unsigned long Delta () const

  Delta returns the time elapsed between the last Tick and the last Tock.
- double Scale () const

  Scale returns the relative lateness or eagerness of the Timer, Relative to a benchmark or Key Frame Rate (The default is 60FPS, or 16667 msec.)

Private Attributes

- struct timeval **_T1**
- struct timeval **_T2**
- unsigned long **_delta**
- double **_scale**

4.15.1 Detailed Description

Definition at line 6 of file Timer.hpp.

4.15.2 Member Function Documentation

4.15.2.1 unsigned long Timer::Delta (void) const

Delta returns the time elapsed between the last Tick and the last Tock.

Does not start a new timer.

Returns

Time elapsed in Microseconds, or Nanoseconds if _RT was enabled.

Definition at line 59 of file Timer.cpp.

4.15.2.2 double Timer::Scale (void) const

Scale returns the relative lateness or eagerness of the Timer, Relative to a benchmark or Key Frame Rate (The default is 60FPS, or 16667 msec.)

Returns

A non-zero float that ranges from (0,1) indicating that the program is rendering faster than 60FPS, or from the range [1,+inf) indicating that the program is rendering slower than 60FPS.

Definition at line 72 of file Timer.cpp.

4.15.2.3 unsigned long Timer::Tick (void)

Tick is an alias for Tock.

Ha, Ha, Ha.

Returns

An unsigned long corresponding to how much time has passed since the last Tick. Microseconds normally, Nanoseconds if _RT was enabled.

Definition at line 29 of file Timer.cpp.

4.15.2.4 unsigned long Timer::Tock (void)

Tock returns the time elapsed since the last Tock.

Returns

An unsigned long corresponding to how much time has passed since the last Tock. Microseconds normally, Nanoseconds if _RT was enabled.

Definition at line 39 of file Timer.cpp.

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

- Timer.hpp
- Timer.cpp

4.16 TransCache Class Reference

Public Member Functions

- void **PTM** (const Angel::mat4 &ptm_in, bool postmult=true)
- const Angel::mat4 & **PTM** (void) const
- const Angel::mat4 & **CTM** (void) const
- const Angel::mat4 & **OTM** (void) const
- void **CalcCTM** (bool postmult=true)

Public Attributes

- TransMat **PreOffset**
- RotMat **PreRotation**
- ScaleMat **scale**
- RotMat **rotation**
- TransMat **offset**
- RotMat **orbit**
- TransMat **displacement**

Private Attributes

- Angel::mat4 **ptm**
- Angel::mat4 **ctm**
- Angel::mat4 **otm**

4.16.1 Detailed Description

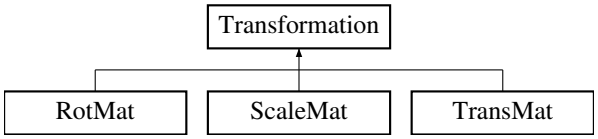
Definition at line 4 of file TransCache.hpp.

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

- TransCache.hpp
- TransCache.cpp

4.17 Transformation Class Reference

Inheritance diagram for Transformation:



Public Member Functions

- const Angel::mat4 & **Matrix** (void) const
- Angel::mat4 **operator*** (const Angel::mat4 &rhs) const
- Angel::mat4 **operator*** (const Transformation &rhs) const

Protected Attributes

- Angel::mat4 **mat**

4.17.1 Detailed Description

Definition at line 8 of file Transformation.hpp.

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

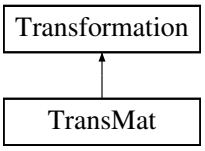
- Transformation.hpp
- Transformation.cpp

4.18 TransMat Class Reference

Translations.

`#include <Transformation.hpp>`

Inheritance diagram for TransMat:



Public Member Functions

- const TransMat & **SetX** (const float x)
- const TransMat & **SetY** (const float y)
- const TransMat & **SetZ** (const float z)
- const TransMat & **Set** (const float x, const float y, const float z)
- const TransMat & **Set** (const Angel::vec3 &arg)
- const TransMat & **Delta** (const float x, const float y, const float z)
- const TransMat & **Delta** (const Angel::vec3 &arg)
- const Angel::mat4 & **Matrix** (void) const
- Angel::mat4 **operator*** (const Angel::mat4 &rhs) const
- Angel::mat4 **operator*** (const Transformation &rhs) const

Protected Attributes

- Angel::mat4 **mat**

4.18.1 Detailed Description

Translations.

Definition at line 47 of file Transformation.hpp.

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

- Transformation.hpp
- Transformation.cpp

4.19 Angel::vec2 Struct Reference

Public Member Functions

- **vec2** (GLfloat s=GLfloat(0.0))
- **vec2** (GLfloat x, GLfloat y)
- **vec2** (const vec2 &v)
- GLfloat & **operator[]** (int i)
- const GLfloat **operator[]** (int i) const
- vec2 **operator-** () const
- vec2 **operator+** (const vec2 &v) const
- vec2 **operator-** (const vec2 &v) const
- vec2 **operator*** (const GLfloat s) const
- vec2 **operator*** (const vec2 &v) const
- vec2 **operator/** (const GLfloat s) const
- vec2 & **operator+=** (const vec2 &v)
- vec2 & **operator-=** (const vec2 &v)
- vec2 & **operator*=** (const GLfloat s)
- vec2 & **operator*=** (const vec2 &v)
- vec2 & **operator/=** (const GLfloat s)
- **operator const GLfloat** * () const
- **operator GLfloat** * ()

Public Attributes

- GLfloat **x**
- GLfloat **y**

Friends

- vec2 **operator*** (const GLfloat s, const vec2 &v)
- std::ostream & **operator<<** (std::ostream &os, const vec2 &v)
- std::istream & **operator>>** (std::istream &is, vec2 &v)

4.19.1 Detailed Description

Definition at line 16 of file vec.hpp.

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

- vec.hpp
- vec.cpp

4.20 Angel::vec3 Struct Reference

Public Member Functions

- **vec3** (GLfloat s=GLfloat(0.0))
- **vec3** (GLfloat x, GLfloat y, GLfloat z)
- **vec3** (const vec3 &v)
- **vec3** (const vec2 &v, const float f)
- GLfloat & **operator[]** (int i)

- const GLfloat **operator[]** (int i) const
- vec3 **operator-** () const
- vec3 **operator+** (const vec3 &v) const
- vec3 **operator-** (const vec3 &v) const
- vec3 **operator*** (const GLfloat s) const
- vec3 **operator*** (const vec3 &v) const
- vec3 **operator/** (const GLfloat s) const
- vec3 & **operator+=** (const vec3 &v)
- vec3 & **operator-=** (const vec3 &v)
- vec3 & **operator*=** (const GLfloat s)
- vec3 & **operator*=** (const vec3 &v)
- vec3 & **operator/=** (const GLfloat s)
- **operator const GLfloat** * () const
- **operator GLfloat** * ()

Public Attributes

- GLfloat **x**
- GLfloat **y**
- GLfloat **z**

Friends

- vec3 **operator*** (const GLfloat s, const vec3 &v)
- std::ostream & **operator<<** (std::ostream &os, const vec3 &v)
- std::istream & **operator>>** (std::istream &is, vec3 &v)

4.20.1 Detailed Description

Definition at line 61 of file vec.hpp.

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

- vec.hpp
- vec.cpp

4.21 Angel::vec4 Struct Reference

Public Member Functions

- **vec4** (GLfloat s=GLfloat(0.0))
- **vec4** (GLfloat x, GLfloat y, GLfloat z, GLfloat w)
- **vec4** (const vec4 &v)
- **vec4** (const vec3 &v, const float w=1.0)
- **vec4** (const vec2 &v, const float z, const float w)
- GLfloat & **operator[]** (int i)
- const GLfloat **operator[]** (int i) const
- vec4 **operator-** () const
- vec4 **operator+** (const vec4 &v) const
- vec4 **operator-** (const vec4 &v) const
- vec4 **operator*** (const GLfloat s) const
- vec4 **operator*** (const vec4 &v) const

- vec4 **operator/** (const GLfloat s) const
- vec4 & **operator+=** (const vec4 &v)
- vec4 & **operator-=** (const vec4 &v)
- vec4 & **operator∗=** (const GLfloat s)
- vec4 & **operator∗=** (const vec4 &v)
- vec4 & **operator/=** (const GLfloat s)
- **operator const GLfloat** ∗ () const
- **operator GLfloat** ∗ ()

Public Attributes

- GLfloat **x**
- GLfloat **y**
- GLfloat **z**
- GLfloat **w**

Friends

- vec4 **operator∗** (const GLfloat s, const vec4 &v)
- std::ostream & **operator<<** (std::ostream &os, const vec4 &v)
- std::istream & **operator>>** (std::istream &is, vec4 &v)

4.21.1 Detailed Description

Definition at line 111 of file vec.hpp.

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

- vec.hpp
- vec.cpp

4.22 wiiPollData Struct Reference

Public Attributes

- Angel::vec3 **bb_magnitudes**
- Angel::vec3 **wr_thetas**
- Angel::vec3 **wr_rates**
- bool **Reset_Camera**

4.22.1 Detailed Description

Definition at line 7 of file WiiUtil.h.

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

- WiiUtil.h

Chapter 5

File Documentation

5.1 Camera.cpp File Reference

Implementation for the Camera class.

```
#include <stdexcept>
#include <iostream>
#include "mat.hpp"
#include "vec.hpp"
#include "Camera.hpp"
#include "globals.h"
#include "Timer.hpp"
```

Macros

- #define ROTATE_OFFSET(V) (V ∗ _ctm.orbit.Matrix())

  ROTATE_OFFSET is a macro which is used to normalize the six camera motion directions with respect to the current camera rotation.

5.1.1 Detailed Description

Implementation for the Camera class.

Author

John Huston

Authors

John Huston, Nicholas StPierre, Chris Compton

Date

2012-12-20

Definition in file Camera.cpp.

5.1.2 Macro Definition Documentation

5.1.2.1 #define ROTATE_OFFSET(*V*) (V * _ctm.orbit.Matrix())

ROTATE_OFFSET is a macro which is used to normalize the six camera motion directions with respect to the current camera rotation.

It is used in heave(), sway() and surge().

Parameters

V	a vec4 representing the movement offset vector.

Returns

A rotated vec4.

Definition at line 184 of file Camera.cpp.

5.2 Camera.hpp File Reference

Header for the Camera class.

```
#include <string>
#include "mat.hpp"
#include "vec.hpp"
#include "Object.hpp"
```

Classes

- class Camera

  The Camera class represents a logical camera in a model view, which posesses a current viewing angle and an absolute position in space as its state.

5.2.1 Detailed Description

Header for the Camera class.

Author

John Huston

Authors

John Huston, Nicholas StPierre, Chris Compton

Date

2013-03-13

Definition in file Camera.hpp.

5.3 Cameras.cpp File Reference

Implementation for the Cameras class, which is a container for Camera objects.

```
#include <cmath>
#include <vector>
#include <stdexcept>
#include "Camera.hpp"
#include "Cameras.hpp"
#include "globals.h"
```

5.3.1 Detailed Description

Implementation for the Cameras class, which is a container for Camera objects.

Author

John Huston

Authors

John Huston, Nicholas StPierre, Chris Compton

Date

2012-12-04

Definition in file Cameras.cpp.

5.4 Cameras.hpp File Reference

Header for the 'Cameras' class, a collection of Camera objects.

```
#include <vector>
#include "Camera.hpp"
#include "Scene.hpp"
```

Classes

- class Cameras

  The Cameras class represents a group of logical cameras for a model view. Each camera possesses its own current viewing angle, and an absolute position in space.

5.4.1 Detailed Description

Header for the 'Cameras' class, a collection of Camera objects.

Author

John Huston

Authors

John Huston, Nicholas StPierre, Chris Compton

Date

2012-12-04

Definition in file Cameras.hpp.

5.5 ds.cpp File Reference

Dual-shader demo.

```
#include <SOIL.h>
#include "globals.h"
#include "platform.h"
#include "Camera.hpp"
#include "Cameras.hpp"
#include "Screen.hpp"
#include "Object.hpp"
#include "Timer.hpp"
#include "Scene.hpp"
#include "Engine.hpp"
#include "model.hpp"
#include "InitShader.hpp"
#include "glut_callbacks.h"
```

Functions

- void init ()

  Initialization: load and compile shaders, initialize camera(s), load models.
- void cleanup (void)

  Cleans up our scene graph.
- void draw (void)

  Implementation of drawing the display with regards to a single viewport.
- void display (void)

  Display/Render the entire screen.
- void idle (void)

  Compute time since last idle, update camera positions, redisplay.
- int main (int argc, char **argv)

  This is a dual-shader demo! It looks very simple, but it illustrates quickly and effectively how to use two shaders.

5.5.1 Detailed Description

Dual-shader demo.

Author

John Huston

Authors

John Huston, Greg Giannone

Date

2013-02-20

Work in progress! Based loosely on Ed Angel's tutorials.

Definition in file ds.cpp.

5.5.2 Function Documentation

5.5.2.1 int main (int *argc,* char ** *argv*)

This is a dual-shader demo! It looks very simple, but it illustrates quickly and effectively how to use two shaders.

Parameters

argc	Not used.
argv	Not used.

Returns

EXIT_SUCCESS.

Definition at line 139 of file ds.cpp.

5.6 Engine.cpp File Reference

Implementation for the Engine class.

```
#include <stdexcept>
#include <string>
#include <map>
#include "Engine.hpp"
#include "Cameras.hpp"
#include "Scene.hpp"
#include "Screen.hpp"
```

5.6.1 Detailed Description

Implementation for the Engine class.

Author

John Huston

Authors

https://github.com/UMLComputerGraphics

Date

2013-03-13

Definition in file Engine.cpp.

5.7 Engine.hpp File Reference

Header for the Engine class.

```
#include "Cameras.hpp"
#include "Scene.hpp"
#include "Screen.hpp"
#include <string>
#include <map>
```

Classes

- class Engine

  The Engine class is a singleton-style class which helps keep track of instances of important objects (for Cameras, Objects, etc) as well as some settings and variables that would otherwise clog up global namespace.

Typedefs

- typedef std::map< std::string, bool > SettingsMap

  An alias for the type used by the Settings Map.

5.7.1 Detailed Description

Header for the Engine class.

Author

John Huston

Authors

https://github.com/UMLComputerGraphics

Date

2013-03-13

Definition in file Engine.hpp.

5.8 globals.h File Reference

Useful global constants, macros, debugging utilities and preprocessor settings.

```
#include <cmath>
```

Macros

- #define DEGREES_TO_RADIANS (M_PI/180)

  A constant factor of conversion for Degrees to Radians.
- #define SQRT2 (1.41421356237)

A constant for the square root of 2.

- #define SQRT3 (1.73205080757)

  A constant for the square root of 3.

- #define POW5(X) ((X)∗(X)∗(X)∗(X)∗(X))

  POW5(X) returns X^5.

- #define DEBUG false

  DEBUG Controls whether or not extra DEBUG messages are printed out.

- #define DEBUG_MOTION (false)

  DEBUG_MOTION controls some additional debug messages for acceleration/velocity.

Variables

- const float DIVIDE_BY_ZERO_TOLERANCE = float(1.0e-07)

  How close can a number be to zero before it should be considered 'zero'?

- static const bool POSTMULT = true

  defines if we are, or are not using a Post-Multiplication system.

- static const bool PREMULT = false

  defines if we are, or are not using a Pre-Multiplication system.

5.8.1 Detailed Description

Useful global constants, macros, debugging utilities and preprocessor settings.

Author

John Huston

Date

2013-03-13

Definition in file globals.h.

5.8.2 Macro Definition Documentation

5.8.2.1 #define DEGREES_TO_RADIANS (M_PI/180)

A constant factor of conversion for Degrees to Radians.

Definition at line 14 of file globals.h.

5.8.2.2 #define POW5(*X*) ((X)∗(X)∗(X)∗(X)∗(X))

POW5(X) returns X^5.

It's quicker than pow(x,5)!

Definition at line 20 of file globals.h.

5.8.2.3 #define SQRT2 (1.41421356237)

A constant for the square root of 2.

Definition at line 16 of file globals.h.

5.8.2.4 #define SQRT3 (1.73205080757)

A constant for the square root of 3.

Definition at line 18 of file globals.h.

5.8.3 Variable Documentation

5.8.3.1 const bool POSTMULT = true [static]

defines if we are, or are not using a Post-Multiplication system.

Definition at line 43 of file globals.h.

5.8.3.2 const bool PREMULT = false [static]

defines if we are, or are not using a Pre-Multiplication system.

Definition at line 45 of file globals.h.

5.9 glut_callbacks.cpp File Reference

glut_callbacks provides function declarations for a set of functions commonly used across multiple binaries for keyboard, mouse and other GLUT callback functions.

```
#include "globals.h"
#include "Camera.hpp"
#include "Scene.hpp"
#include "Screen.hpp"
#include "Engine.hpp"
#include <sstream>
```

Functions

- void keylift (unsigned char key, int x, int y)

  keylift is registered as a GLUT callback for when a user releases a depressed key.
- void keyboard (unsigned char key, int x, int y)

  keyboard is a callback registered with GLUT.
- void keyboard_ctrl (int key, int x, int y)

  keyboard_ctrl is registered as a GLUT callback.
- void mouse (int button, int state, int x, int y)

  mouse is registered as a GLUT callback.
- void mouseroll (int x, int y)

  mouseroll is registered as a GLUT callback.
- void mouselook (int x, int y)

  mouselook is registered as a GLUT callback.
- void resizeEvent (int width, int height)

  resizeEvent is registered as a glut callback for when the screen is resized.

5.9.1 Detailed Description

glut_callbacks provides function declarations for a set of functions commonly used across multiple binaries for keyboard, mouse and other GLUT callback functions.

Author

John Huston

Authors

John Huston, Nick St.Pierre, Chris Compton

Date

2013-03-13

Definition in file glut_callbacks.cpp.

5.9.2 Function Documentation

5.9.2.1 void keyboard (unsigned char *key*, int *x*, int *y*)

keyboard is a callback registered with GLUT.

It handles (surprise!) keyboard input.

Parameters

key	The key pressed by the user.
x	The x coordinate of the mouse when the key was pressed.
y	The y coordinate of the mouse when the key was pressed.

Definition at line 66 of file glut_callbacks.cpp.

5.9.2.2 void keyboard_ctrl (int *key*, int *x*, int *y*)

keyboard_ctrl is registered as a GLUT callback.

It is responsible for catching when special keys are pressed.

Parameters

key	The key pressed.
x	The x coordinate of the mouse when the key was pressed.
y	The y coordinate of the mouse when the key was pressed.

Definition at line 167 of file glut_callbacks.cpp.

5.9.2.3 void keylift (unsigned char *key*, int *x*, int *y*)

keylift is registered as a GLUT callback for when a user releases a depressed key.

Parameters

key	The key that was lifted.
x	The x coordinate of the mouse at the time the key was released.
y	The y coordinate of the mouse at the time the key was released.

Definition at line 29 of file glut_callbacks.cpp.

5.9.2.4 void mouse (int *button*, int *state*, int *x*, int *y*)

mouse is registered as a GLUT callback.

It handles input from, primarily, the scrollwheel.

Parameters

button	The mouse button being pressed.
state	the state of the aforementioned mouse button.
x	the x coordinate of the mouse.
y	the y coordinate of the mouse.

Definition at line 219 of file glut_callbacks.cpp.

5.9.2.5 void mouselook (int *x*, int *y*)

mouselook is registered as a GLUT callback.

mouselook implements FPS-like controls where the camera moves proportional to the direction of the mouse.

Parameters

x	the x coordinate of the mouse pointer.
y	the y coordinate of the mouse pointer.

Definition at line 265 of file glut_callbacks.cpp.

5.9.2.6 void mouseroll (int *x*, int *y*)

mouseroll is registered as a GLUT callback.

mouseroll is called when the mouse is moved while a button is depressed. It is used here to implement barrel-rolls while left-clicking.

Parameters

x	the x coordinate of the mouse pointer.
y	the y coordinate of the mouse pointer.

Definition at line 245 of file glut_callbacks.cpp.

5.9.2.7 void resizeEvent (int *width*, int *height*)

resizeEvent is registered as a glut callback for when the screen is resized.

It instructs the screen object of the new size, which informs all of the children cameras to recompute their aspect ratios, viewport positions, and so on.

We also warp the pointer to the center of the screen, for compatibility with mouselook(void).

Parameters

width	The new width of the window.
height	The new height of the window.

Returns

void.

Definition at line 299 of file glut_callbacks.cpp.

5.10 glut_callbacks.h File Reference

glut_callbacks.h provides function declarations for a set of functions commonly used across multiple binaries for keyboard, mouse and other GLUT callback functions.

Functions

- void keylift (unsigned char key, int x, int y)

  keylift is registered as a GLUT callback for when a user releases a depressed key.
- void keyboard (unsigned char key, int x, int y)

  keyboard is a callback registered with GLUT.
- void keyboard_ctrl (int key, int x, int y)

  keyboard_ctrl is registered as a GLUT callback.
- void mouse (int button, int state, int x, int y)

  mouse is registered as a GLUT callback.
- void mouseroll (int x, int y)

  mouseroll is registered as a GLUT callback.
- void mouselook (int x, int y)

  mouselook is registered as a GLUT callback.
- void resizeEvent (int width, int height)

  resizeEvent is registered as a glut callback for when the screen is resized.

5.10.1 Detailed Description

glut_callbacks.h provides function declarations for a set of functions commonly used across multiple binaries for keyboard, mouse and other GLUT callback functions.

Author

John Huston

Authors

John Huston, Nick St.Pierre, Chris Compton

Date

2013-03-13

Definition in file glut_callbacks.h.

5.10.2 Function Documentation

5.10.2.1 void keyboard (unsigned char *key*, int *x*, int *y*)

keyboard is a callback registered with GLUT.

It handles (surprise!) keyboard input.

Parameters

key	The key pressed by the user.
x	The x coordinate of the mouse when the key was pressed.
y	The y coordinate of the mouse when the key was pressed.

Definition at line 66 of file glut_callbacks.cpp.

5.10.2.2 void keyboard_ctrl (int *key*, int *x*, int *y*)

keyboard_ctrl is registered as a GLUT callback.

It is responsible for catching when special keys are pressed.

Parameters

key	The key pressed.
x	The x coordinate of the mouse when the key was pressed.
y	The y coordinate of the mouse when the key was pressed.

Definition at line 167 of file glut_callbacks.cpp.

5.10.2.3 void keylift (unsigned char *key*, int *x*, int *y*)

keylift is registered as a GLUT callback for when a user releases a depressed key.

Parameters

key	The key that was lifted.
x	The x coordinate of the mouse at the time the key was released.
y	The y coordinate of the mouse at the time the key was released.

Definition at line 29 of file glut_callbacks.cpp.

5.10.2.4 void mouse (int *button*, int *state*, int *x*, int *y*)

mouse is registered as a GLUT callback.

It handles input from, primarily, the scrollwheel.

Parameters

button	The mouse button being pressed.
state	the state of the aforementioned mouse button.
x	the x coordinate of the mouse.
y	the y coordinate of the mouse.

Definition at line 219 of file glut_callbacks.cpp.

5.10.2.5 void mouselook (int *x,* int *y*)

mouselook is registered as a GLUT callback.

mouselook implements FPS-like controls where the camera moves proportional to the direction of the mouse.

Parameters

x	the x coordinate of the mouse pointer.
y	the y coordinate of the mouse pointer.

Definition at line 265 of file glut_callbacks.cpp.

5.10.2.6 void mouseroll (int *x,* int *y*)

mouseroll is registered as a GLUT callback.

mouseroll is called when the mouse is moved while a button is depressed. It is used here to implement barrel-rolls while left-clicking.

Parameters

x	the x coordinate of the mouse pointer.
y	the y coordinate of the mouse pointer.

Definition at line 245 of file glut_callbacks.cpp.

5.10.2.7 void resizeEvent (int *width,* int *height*)

resizeEvent is registered as a glut callback for when the screen is resized.

It instructs the screen object of the new size, which informs all of the children cameras to recompute their aspect ratios, viewport positions, and so on.

We also warp the pointer to the center of the screen, for compatibility with mouselook(void).

Parameters

width	The new width of the window.
height	The new height of the window.

Returns

void.

Definition at line 299 of file glut_callbacks.cpp.

5.11 InitShader.cpp File Reference

Provides a wrapper utility for quickly linking against glsl programs.

```
#include <cstdio>
#include <iostream>
#include "platform.h"
#include "InitShader.hpp"
```

Macros

- #define GEOMETRY_VERTICES_OUT_EXT 0x8DDA

  GEOMETRY_VERTICES_OUT_EXT is a Magic OpenGL constant.
- #define GL_GEOMETRY_SHADER 0x8DD9

  GEOMETRY_VERTICES_OUT_EXT is a Magic OpenGL constant.

Functions

- static char * **Angel::readShaderSource** (const char *shaderFile)

  Read in a shader file into a NULL-terminated string.
- GLuint **Angel::InitShader** (const char *vShaderFile, const char *fShaderFile)

  InitShader takes two shader sourcefiles and compiles them into a shader program.
- GLuint **Angel::InitShader** (const char *vShaderFile, const char *gShaderFile, const char *fShaderFile)

  InitShader takes three shader sourcefiles and compiles them into a shader program.

5.11.1 Detailed Description

Provides a wrapper utility for quickly linking against glsl programs.

Authors

Ed Angel, Nick St.Pierre

Date

2013-03-13

Definition in file InitShader.cpp.

5.11.2 Macro Definition Documentation

5.11.2.1 #define GEOMETRY_VERTICES_OUT_EXT 0x8DDA

GEOMETRY_VERTICES_OUT_EXT is a Magic OpenGL constant.

On some systems, we might need to define this manually. It is normally provided by OpenGL directly. FIXME: This seems hacky!

Definition at line 22 of file InitShader.cpp.

5.11.2.2 #define GL_GEOMETRY_SHADER 0x8DD9

GEOMETRY_VERTICES_OUT_EXT is a Magic OpenGL constant.

On some systems, we might need to define this manually. It is normally provided by OpenGL directly. FIXME: This seems hacky!

Definition at line 33 of file InitShader.cpp.

5.12 InitShader.hpp File Reference

Provides a wrapper utility for quickly linking against glsl programs.

Functions

- GLuint **Angel::InitShader** (const char ∗vShaderFile, const char ∗fShaderFile)

  InitShader takes two shader sourcefiles and compiles them into a shader program.
- GLuint **Angel::InitShader** (const char ∗vShaderFile, const char ∗gShaderFile, const char ∗fShaderFile)

  InitShader takes three shader sourcefiles and compiles them into a shader program.

5.12.1 Detailed Description

Provides a wrapper utility for quickly linking against glsl programs.

Authors

Ed Angel, Nick St.Pierre

Date

2013-03-13

Definition in file InitShader.hpp.

5.13 KinectInator.cpp File Reference

TODO FIXME.

```
#include "KinectInator.hpp"
```

Macros

- #define **CHECK_RC**(nRetVal, what)

Functions

- XnBool **fileExists** (const char ∗fn)
- void **printhead** (int user, double x, double y, double z)
- void **noop** (int user, double x, double y, double z)

Variables

- boost::function< void XN_CALLBACK_TYPE(xn::UserGenerator &, XnUserID, void ∗)> **TiemSpelchk::_new_user**
- boost::function< void XN_CALLBACK_TYPE(xn::UserGenerator &, XnUserID, void ∗)> **TiemSpelchk::_lost_user**
- boost::function< void XN_CALLBACK_TYPE(xn::PoseDetectionCapability &, const XnChar ∗, XnUserID, void ∗)> **TiemSpelchk::_pose**
- boost::function< void XN_CALLBACK_TYPE(xn::SkeletonCapability &, XnUserID, void ∗)> **TiemSpelchk::_cal_start**

- boost::function< void XN_CALLBACK_TYPE(xn::SkeletonCapability &, XnUserID, XnCalibrationStatus, void *)> **TiemSpelchk::_cal_complete**

5.13.1 Detailed Description

TODO FIXME.

Date

2013-03-13

Authors

PrimeSense Ltd, Eric McCann

Definition in file KinectInator.cpp.

5.13.2 Macro Definition Documentation

5.13.2.1 #define CHECK_RC(*nRetVal*, *what*)

Value:

```
if (nRetVal != XN_STATUS_OK)                    \
{                                               \
    printf("%s failed: %s\n", what, xnGetStatusString(nRetVal));    \
    return nRetVal;                             \
}
```

Definition at line 104 of file KinectInator.cpp.

5.14 KinectInator.hpp File Reference

TODO FIXME.

```
#include <XnCppWrapper.h>
#include <boost/thread/thread.hpp>
#include <boost/thread/mutex.hpp>
#include <boost/function.hpp>
#include <iostream>
#include <cfloat>
```

Classes

- class TiemSpelchk::Lurn2SpielNub

Macros

- #define **__OPENNIFTW**
- #define **SAMPLE_XML_PATH** "OpenNIConfig.xml"
- #define **SAMPLE_XML_PATH_LOCAL** "OpenNIConfig.xml"
- #define **MAX_NUM_USERS** 15

Functions

- void **printhead** (int, double, double, double)
- void **noop** (int, double, double, double)
- void **noopint** (int)

5.14.1 Detailed Description

TODO FIXME.

Date

2013-03-13

Author

PrimeSense Ltd.

Authors

PrimeSense Ltd., Eric McCann

Definition in file KinectInator.hpp.

5.15 mat.cpp File Reference

Implementation for the mat2, mat3, and mat4 classes.

```
#include "platform.h"
#include "vec.hpp"
#include "mat.hpp"
#include <cmath>
#include "globals.h"
```

Functions

- mat2 **Angel::operator**∗ (const GLfloat s, const mat2 &m)
- std::ostream & **Angel::operator**<< (std::ostream &os, const mat2 &m)
- std::istream & **Angel::operator**>> (std::istream &is, mat2 &m)
- mat2 **Angel::matrixCompMult** (const mat2 &A, const mat2 &B)
- mat2 **Angel::transpose** (const mat2 &A)
- mat3 **Angel::operator**∗ (const GLfloat s, const mat3 &m)
- std::ostream & **Angel::operator**<< (std::ostream &os, const mat3 &m)
- std::istream & **Angel::operator**>> (std::istream &is, mat3 &m)
- mat3 **Angel::matrixCompMult** (const mat3 &A, const mat3 &B)
- mat3 **Angel::transpose** (const mat3 &A)
- mat4 **Angel::operator**∗ (const GLfloat s, const mat4 &m)
- vec4 **Angel::operator**∗ (const vec4 &v, const mat4 &_m)
- std::ostream & **Angel::operator**<< (std::ostream &os, const mat4 &m)
- std::istream & **Angel::operator**>> (std::istream &is, mat4 &m)
- mat4 **Angel::matrixCompMult** (const mat4 &A, const mat4 &B)
- mat4 **Angel::transpose** (const mat4 &A)
- mat4 **Angel::RotateX** (const GLfloat theta)

- mat4 **Angel::RotateY** (const GLfloat theta)
- mat4 **Angel::RotateZ** (const GLfloat theta)
- mat4 **Angel::Translate** (const GLfloat x, const GLfloat y, const GLfloat z)
- mat4 **Angel::Translate** (const vec3 &v)
- mat4 **Angel::Translate** (const vec4 &v)
- mat4 **Angel::Scale** (const GLfloat x, const GLfloat y, const GLfloat z)
- mat4 **Angel::Scale** (const vec3 &v)
- mat4 **Angel::Ortho** (const GLfloat left, const GLfloat right, const GLfloat bottom, const GLfloat top, const GLfloat zNear, const GLfloat zFar)
- mat4 **Angel::Ortho2D** (const GLfloat left, const GLfloat right, const GLfloat bottom, const GLfloat top)
- mat4 **Angel::Frustum** (const GLfloat left, const GLfloat right, const GLfloat bottom, const GLfloat top, const GLfloat zNear, const GLfloat zFar)
- mat4 **Angel::Perspective** (const GLfloat fovy, const GLfloat aspect, const GLfloat zNear, const GLfloat zFar)
- mat4 **Angel::LookAt** (const vec4 &eye, const vec4 &at, const vec4 &up)

5.15.1 Detailed Description

Implementation for the mat2, mat3, and mat4 classes.

Author

Ed Angel

Date

2012-12-04

Modified heavily from code available from Ed Angel's website, http://www.cs.unm.edu/~angel/BOOK/-INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/ Published from his book, Interactive Computer Graphics A Top-Down Approach with OpenGL, Sixth Edition Addison-Wesley 2012

Definition in file mat.cpp.

5.16 mat.hpp File Reference

Headers for the mat2, mat3, and mat4 classes and related utilities.

```
#include <cstdio>
#include "vec.hpp"
```

Classes

- class Angel::mat2

  mat2 - 2D square matrix.
- class Angel::mat3

  mat3 - 3D square matrix.
- class Angel::mat4

  mat4 - 4D square matrix.

Macros

- #define **Error**(str)

Functions

- mat2 **Angel::matrixCompMult** (const mat2 &A, const mat2 &B)
- mat2 **Angel::transpose** (const mat2 &A)
- mat3 **Angel::matrixCompMult** (const mat3 &A, const mat3 &B)
- mat3 **Angel::transpose** (const mat3 &A)
- mat4 **Angel::matrixCompMult** (const mat4 &A, const mat4 &B)
- mat4 **Angel::transpose** (const mat4 &A)
- mat4 **Angel::RotateX** (const GLfloat theta)
- mat4 **Angel::RotateY** (const GLfloat theta)
- mat4 **Angel::RotateZ** (const GLfloat theta)
- mat4 **Angel::Translate** (const GLfloat x, const GLfloat y, const GLfloat z)
- mat4 **Angel::Translate** (const vec3 &v)
- mat4 **Angel::Translate** (const vec4 &v)
- mat4 **Angel::Scale** (const GLfloat x, const GLfloat y, const GLfloat z)
- mat4 **Angel::Scale** (const vec3 &v)
- mat4 **Angel::Ortho** (const GLfloat left, const GLfloat right, const GLfloat bottom, const GLfloat top, const GLfloat zNear, const GLfloat zFar)
- mat4 **Angel::Ortho2D** (const GLfloat left, const GLfloat right, const GLfloat bottom, const GLfloat top)
- mat4 **Angel::Frustum** (const GLfloat left, const GLfloat right, const GLfloat bottom, const GLfloat top, const GLfloat zNear, const GLfloat zFar)
- mat4 **Angel::Perspective** (const GLfloat fovy, const GLfloat aspect, const GLfloat zNear, const GLfloat zFar)
- mat4 **Angel::LookAt** (const vec4 &eye, const vec4 &at, const vec4 &up)

5.16.1 Detailed Description

Headers for the mat2, mat3, and mat4 classes and related utilities.

Author

Ed Angel

Authors

Ed Angel, John Huston

Date

2012-12-04

Modified from code available from Ed Angel's website, http://www.cs.unm.edu/~angel/BOOK/INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/ Published from his book, Interactive Computer Graphics A Top-Down Approach with OpenGL, Sixth Edition Addison-Wesley 2012

Definition in file mat.hpp.

5.16.2 Macro Definition Documentation

5.16.2.1 #define Error(*str*)

Value:

```
do {                \
    std::cerr << "[" __FILE__ ":" << __LINE__ << "] "   \
          << str << std::endl; } while(0)
```

Definition at line 206 of file mat.hpp.

5.17 model.cpp File Reference

Functions related to constructing simple geometry.

```
#include "globals.h"
#include "Object.hpp"
#include "Timer.hpp"
#include "vec.hpp"
#include <cmath>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
```

Macros

- #define QUAD(A, B, C, D)

  A macro to help quickly call quad() with correct parameters.
- #define OFFSET_AT(X, Z) ((X)∗S+(Z))

  Gives the one-dimensional index offset for a buffer given the terrain's X,Z coordinates.
- #define VERTEX_AT(X, Z) (vec.at(OFFSET_AT(X,Z)))

  Gives the vertex for the terrain data at (X,Z).
- #define HEIGHT_AT(X, Z) (VERTEX_AT(X,Z).y)

  Returns the height as a float for these terrain coordinates.
- #define COLOR_AT(X, Z) (col.at(OFFSET_AT(X,Z)))

  Returns the color vector for the terrain data at (X,Z).
- #define TEX_UV_AT(X, Z) (txy.at(OFFSET_AT(X,Z)))

  Returns the Texture UV (vec2) for the terrain at (X,Z).
- #define NORMAL_AT(X, Z) (nor.at(OFFSET_AT(X,Z)))

  Returns the Normal vec3 for the terrain at (X,Z).

Typedefs

- typedef Angel::vec4 color4

  Simple alias of Angel::vec4 to emphasize semantic meaning.
- typedef Angel::vec4 point4

  Simple alias of Angel::vec4 to emphasize semantic meaning.

Functions

- void createPoint (Object ∗obj, point4 const &the_point, color4 const &the_color, vec3 const &the_normal)

  Adds another vertex to the specified object.
- void triangle (Object ∗obj, const point4 &a, const point4 &b, const point4 &c, const int color)

  Creates a triangle primitive from three spatial coordinates.
- point4 unit (const point4 &p)

  Ed Angel utility function: Used to normalize a vector.
- void divideTriangle (Object ∗obj, const point4 &a, const point4 &b, const point4 &c, int timesToRecurse, int color)

  Used in building the Sierpinski gasket: Takes the coordinate for a triangle and splits it into several smaller triangles.
- void tetra (Object ∗obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d)

  Creates a tetrahedron using four triangles.

- void sierpinskiPyramid (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d, int count)

  Forms a Sierpinski Pyramid object given four 4D points in space.
- void recursiveModelGen (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d, int timesToRecurse, int color)

  Given a quadrilateral, splits it up into smaller quadrilaterals.
- void sphere (Object *obj)

  Creates a white sphere.
- void quad (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d, const color4 &A, const color4 &B, const color4 &C, const color4 &D)

  Create a quadrilateral from four points and four colors.
- void cube (Object *obj, const GLfloat &size, const color4 colors[8])

  Create a cube of a given size fixed at the origin, using the eight colors specified.
- void colorCube (Object *obj, GLfloat size)

  Creates a cube of a given size fixed at the origin, using all eight primary colors.
- float randFloat (void)

  Return a randomized float value!
- double jitter (double H)

  Returns a random float between [-H,H].
- vec3 calcNormal (point4 &a, point4 &b, point4 &c)

  Calculate the vector normal to the triangle formed by three points.
- double landGen (Object *obj, int N, float H)

  Use the diamond-square terrain generation algorithm to generate a triangle strip that resembles terrain with oceans, mountains, etc.
- void makeAgua (Object *land_obj, Object *agua_obj)

  Adds a blue quadrilateral to an already-generated terrain object to create the appearance of water.

5.17.1 Detailed Description

Functions related to constructing simple geometry.

Authors

Ed Angel, John Huston, Chris Compton, Nick St.Pierre

Date

2013-03-14

Definition in file model.cpp.

5.17.2 Macro Definition Documentation

5.17.2.1 #define QUAD(*A, B, C, D*)

Value:

```
quad( obj, vertices[A], vertices[B], vertices[C], vertices[D],  \
    colors[A], colors[B], colors[C], colors[D] );
```

A macro to help quickly call quad() with correct parameters.

See Also

quad().

Definition at line 268 of file model.cpp.

5.17.3 Typedef Documentation

5.17.3.1 typedef Angel::vec4 color4

Simple alias of Angel::vec4 to emphasize semantic meaning.

Definition at line 16 of file model.cpp.

5.17.3.2 typedef Angel::vec4 point4

Simple alias of Angel::vec4 to emphasize semantic meaning.

Definition at line 18 of file model.cpp.

5.17.4 Function Documentation

5.17.4.1 vec3 calcNormal (point4 & *a*, point4 & *b*, point4 & *c*)

Calculate the vector normal to the triangle formed by three points.

Parameters

a	First vertex.
b	Second vertex.
c	Third vertex.

Returns

The vector normal to the plane formed by the triangle a,b,c.

Definition at line 352 of file model.cpp.

5.17.4.2 void colorCube (Object * *obj*, GLfloat *size*)

Creates a cube of a given size fixed at the origin, using all eight primary colors.

Parameters

obj	The object to add the geometry to.
size	The size of the cube to create.

Definition at line 310 of file model.cpp.

5.17.4.3 void createPoint (Object * *obj*, point4 const & *the_point*, color4 const & *the_color*, vec3 const & *the_normal*)

Adds another vertex to the specified object.

Parameters

obj	The object to add the vertex to.
the_point	The 4d spatial coordinate of the vertex.
the_color	The vec4 specifying the RGBA color value of the vertex.
the_normal	The vec3 that specifies the normal for this vertex.

Definition at line 42 of file model.cpp.

5.17.4.4 void cube (Object * *obj,* const GLfloat & *size,* const color4 *colors[8]*)

Create a cube of a given size fixed at the origin, using the eight colors specified.

Parameters

obj	The object to add the geometry to.
size	The size of the cube to create.
colors	An array of eight colors for the vertices.

Definition at line 280 of file model.cpp.

5.17.4.5 void divideTriangle (Object * *obj,* const point4 & *a,* const point4 & *b,* const point4 & *c,* int *timesToRecurse,* int *color*)

Used in building the Sierpinski gasket: Takes the coordinate for a triangle and splits it into several smaller triangles.

Parameters

obj	The object to add the triangles to.
a	The first spatial coordinate for the triangle.
b	The second spatial coordinate for the triangle.
c	The third spatial coordinate for the triangle.
timesToRecurse	The number of times to subdivide.
color	An index for the color to use for the triangle: { Red, Green, Blue, Yellow, Pink, White }

Definition at line 123 of file model.cpp.

5.17.4.6 double jitter (double *H*)

Returns a random float between [-H,H].

Parameters

H	The range for the random float.

Returns

a random float between [-H,H].

Definition at line 340 of file model.cpp.

5.17.4.7 double landGen (Object * *obj,* int *N,* float *H*)

Use the diamond-square terrain generation algorithm to generate a triangle strip that resembles terrain with oceans, mountains, etc.

Parameters

obj	The object to add the geometry to.
N	The size of the terrain: Will be n^2 x n^2 evenly spaced vertices.
H	The height 'randomness' factor.

Returns

The maximum height actually achieved in this terrain generation.

Definition at line 402 of file model.cpp.

5.17.4.8 void makeAgua (Object * *land_obj*, Object * *agua_obj*)

Adds a blue quadrilateral to an already-generated terrain object to create the appearance of water.

Parameters

land_obj	
agua_obj	

What should the water's height be?

Definition at line 561 of file model.cpp.

5.17.4.9 void quad (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*, const color4 & *A*, const color4 & *B*, const color4 & *C*, const color4 & *D*)

Create a quadrilateral from four points and four colors.

Parameters

obj	The object to add the geometry to.
a	The first spatial point.
b	The second spatial point.
c	The third spatial point.
d	The fourth spatial point.
A	The color of the first point.
B	The color of the second point.
C	The color of the third point.
D	The color of the fourth point.

Definition at line 242 of file model.cpp.

5.17.4.10 float randFloat (void)

Return a randomized float value!

Returns

A random float, just for you!

Definition at line 331 of file model.cpp.

5.17.4.11 void recursiveModelGen (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*, int *timesToRecurse*, int *color*)

Given a quadrilateral, splits it up into smaller quadrilaterals.

Used in the generation of spheres! FIXME: Nick St.Pierre (Documentation!)

Parameters

obj	The object to add the geometry to.
a	The first spatial coordinate.
b	The second spatial coordinate.
c	The third spatial coordinate.

d	The fourth spatial coordinate.
timesToRecurse	The number of subdivisions to make.
color	An index for the color to use for the triangle: { Red, Green, Blue, Yellow, Pink, White }

Definition at line 200 of file model.cpp.

5.17.4.12 void sierpinskiPyramid (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*, int *count*)

Forms a Sierpinski Pyramid object given four 4D points in space.

Parameters

obj	The object to add the geometry to.
a	The first coordinate.
b	The second coordinate.
c	The third coordinate.
d	The fourth coordinate.
count	The number of recursions to perform to construct the gasket.

Definition at line 168 of file model.cpp.

5.17.4.13 void sphere (Object * *obj*)

Creates a white sphere.

Parameters

obj	The object to add the geometry to.

Definition at line 215 of file model.cpp.

5.17.4.14 void tetra (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*)

Creates a tetrahedron using four triangles.

(12 vertices.)

Parameters

obj	The object to add the Tetrahedron to/
a	The first spatial coordinate for the tetrahedron.
b	The second spatial coordinate for the tetrahedron.
c	The third spatial coordinate for the tetrahedron.
d	The fourth spatial coordinate for the tetrahedron.

Definition at line 149 of file model.cpp.

5.17.4.15 void triangle (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const int *color*)

Creates a triangle primitive from three spatial coordinates.

Parameters

obj	The object to add the triangle to.
a	The location of the first vertex.
b	The location of the second vertex.
c	The location of the third vertex.
color	An index for the color to use for the triangle: { Red, Green, Blue, Yellow, Pink, White }

Definition at line 61 of file model.cpp.

5.17.4.16 point4 unit (const point4 & *p*)

Ed Angel utility function: Used to normalize a vector.

TODO: Is this a redundant version of Angel::normalize?

Parameters

p	

Returns

Definition at line 93 of file model.cpp.

5.18 model.hpp File Reference

Headers for Functions related to constructing simple geometry.

```
#include "vec.hpp"
#include "Object.hpp"
```

Typedefs

- typedef Angel::vec4 color4

  Simple alias of Angel::vec4 to emphasize semantic meaning.
- typedef Angel::vec4 point4

  Simple alias of Angel::vec4 to emphasize semantic meaning.

Functions

- void createPoint (Object *obj, point4 const &the_point, color4 const &the_color, vec3 const &the_normal)

  Adds another vertex to the specified object.
- void triangle (Object *obj, const point4 &a, const point4 &b, const point4 &c, const int color)

  Creates a triangle primitive from three spatial coordinates.
- void divideTriangle (Object *obj, const point4 &a, const point4 &b, const point4 &c, int timesToRecurse, int color)

  Used in building the Sierpinski gasket: Takes the coordinate for a triangle and splits it into several smaller triangles.
- void tetra (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d)

  Creates a tetrahedron using four triangles.
- void sierpinskiPyramid (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d, int count)

Forms a Sierpinski Pyramid object given four 4D points in space.

- void recursiveModelGen (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d, int timesToRecurse, int color)

  Given a quadrilateral, splits it up into smaller quadrilaterals.
- void sphere (Object *obj)

  Creates a white sphere.
- void quad (Object *obj, const point4 &a, const point4 &b, const point4 &c, const point4 &d, const color4 &A, const color4 &B, const color4 &C, const color4 &D)

  Create a quadrilateral from four points and four colors.
- void cube (Object *obj, const GLfloat &size, const color4 colors[8])

  Create a cube of a given size fixed at the origin, using the eight colors specified.
- void colorCube (Object *obj, GLfloat size)

  Creates a cube of a given size fixed at the origin, using all eight primary colors.
- double landGen (Object *obj, int N, float H)

  Use the diamond-square terrain generation algorithm to generate a triangle strip that resembles terrain with oceans, mountains, etc.
- void makeAgua (Object *land_obj, Object *agua_obj)

  Adds a blue quadrilateral to an already-generated terrain object to create the appearance of water.

5.18.1 Detailed Description

Headers for Functions related to constructing simple geometry.

Authors

Ed Angel, John Huston, Chris Compton, Nick St.Pierre

Date

2013-03-14

Definition in file model.hpp.

5.18.2 Typedef Documentation

5.18.2.1 typedef Angel::vec4 color4

Simple alias of Angel::vec4 to emphasize semantic meaning.

Definition at line 17 of file model.hpp.

5.18.2.2 typedef Angel::vec4 point4

Simple alias of Angel::vec4 to emphasize semantic meaning.

Definition at line 19 of file model.hpp.

5.18.3 Function Documentation

5.18.3.1 void colorCube (Object * *obj*, GLfloat *size*)

Creates a cube of a given size fixed at the origin, using all eight primary colors.

Parameters

obj	The object to add the geometry to.
size	The size of the cube to create.

Definition at line 310 of file model.cpp.

5.18.3.2 void createPoint (Object * *obj*, point4 const & *the_point*, color4 const & *the_color*, vec3 const & *the_normal*)

Adds another vertex to the specified object.

Parameters

obj	The object to add the vertex to.
the_point	The 4d spatial coordinate of the vertex.
the_color	The vec4 specifying the RGBA color value of the vertex.
the_normal	The vec3 that specifies the normal for this vertex.

Definition at line 42 of file model.cpp.

5.18.3.3 void cube (Object * *obj*, const GLfloat & *size*, const color4 *colors[8]*)

Create a cube of a given size fixed at the origin, using the eight colors specified.

Parameters

obj	The object to add the geometry to.
size	The size of the cube to create.
colors	An array of eight colors for the vertices.

Definition at line 280 of file model.cpp.

5.18.3.4 void divideTriangle (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, int *timesToRecurse*, int *color*)

Used in building the Sierpinski gasket: Takes the coordinate for a triangle and splits it into several smaller triangles.

Parameters

obj	The object to add the triangles to.
a	The first spatial coordinate for the triangle.
b	The second spatial coordinate for the triangle.
c	The third spatial coordinate for the triangle.
timesToRecurse	The number of times to subdivide.
color	An index for the color to use for the triangle: { Red, Green, Blue, Yellow, Pink, White }

Definition at line 123 of file model.cpp.

5.18.3.5 double landGen (Object * *obj*, int *N*, float *H*)

Use the diamond-square terrain generation algorithm to generate a triangle strip that resembles terrain with oceans, mountains, etc.

Parameters

obj	The object to add the geometry to.
N	The size of the terrain: Will be n^2 x n^2 evenly spaced vertices.

H	The height 'randomness' factor.

Returns

The maximum height actually achieved in this terrain generation.

Definition at line 402 of file model.cpp.

5.18.3.6 void makeAgua (Object * *land_obj*, Object * *agua_obj*)

Adds a blue quadrilateral to an already-generated terrain object to create the appearance of water.

Parameters

land_obj	
agua_obj	

What should the water's height be?

Definition at line 561 of file model.cpp.

5.18.3.7 void quad (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*, const color4 & *A*, const color4 & *B*, const color4 & *C*, const color4 & *D*)

Create a quadrilateral from four points and four colors.

Parameters

obj	The object to add the geometry to.
a	The first spatial point.
b	The second spatial point.
c	The third spatial point.
d	The fourth spatial point.
A	The color of the first point.
B	The color of the second point.
C	The color of the third point.
D	The color of the fourth point.

Definition at line 242 of file model.cpp.

5.18.3.8 void recursiveModelGen (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*, int *timesToRecurse*, int *color*)

Given a quadrilateral, splits it up into smaller quadrilaterals.

Used in the generation of spheres! FIXME: Nick St.Pierre (Documentation!)

Parameters

obj	The object to add the geometry to.
a	The first spatial coordinate.
b	The second spatial coordinate.
c	The third spatial coordinate.
d	The fourth spatial coordinate.
timesToRecurse	The number of subdivisions to make.
color	An index for the color to use for the triangle: { Red, Green, Blue, Yellow, Pink, White }

Definition at line 200 of file model.cpp.

5.18.3.9 void sierpinskiPyramid (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*, int *count*)

Forms a Sierpinski Pyramid object given four 4D points in space.

Parameters

obj	The object to add the geometry to.
a	The first coordinate.
b	The second coordinate.
c	The third coordinate.
d	The fourth coordinate.
count	The number of recursions to perform to construct the gasket.

Definition at line 168 of file model.cpp.

5.18.3.10 void sphere (Object * *obj*)

Creates a white sphere.

Parameters

obj	The object to add the geometry to.

Definition at line 215 of file model.cpp.

5.18.3.11 void tetra (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const point4 & *d*)

Creates a tetrahedron using four triangles.

(12 vertices.)

Parameters

obj	The object to add the Tetrahedron to/
a	The first spatial coordinate for the tetrahedron.
b	The second spatial coordinate for the tetrahedron.
c	The third spatial coordinate for the tetrahedron.
d	The fourth spatial coordinate for the tetrahedron.

Definition at line 149 of file model.cpp.

5.18.3.12 void triangle (Object * *obj*, const point4 & *a*, const point4 & *b*, const point4 & *c*, const int *color*)

Creates a triangle primitive from three spatial coordinates.

Parameters

obj	The object to add the triangle to.
a	The location of the first vertex.
b	The location of the second vertex.
c	The location of the third vertex.
color	An index for the color to use for the triangle: { Red, Green, Blue, Yellow, Pink, White }

Definition at line 61 of file model.cpp.

5.19 ObjLoader.cpp File Reference

Implementation for reading in geometry from .OBJ files.

```
#include <vector>
#include <string>
#include <sstream>
#include <fstream>
#include <stdexcept>
#include <cstdio>
#include "vec.hpp"
#include "ObjLoader.hpp"
```

Macros

- #define RAND_FLOAT (rand() / (float) RAND_MAX)
  RAND_FLOAT returns a random float from (0,1).

Functions

- const vector< string > **ObjLoader::split** (const string &str, const char delim)
  Split a string by an arbitrary delimiter.
- vec4 **ObjLoader::parseVertex** (string line)
  Obtain a vertex from a std::string line.
- vec2 **ObjLoader::parseTextureUV** (string line)
  Obtain a 2D TexUV coordinate from a std::string line.
- vec3 **ObjLoader::parseNormal** (string line)
  Obtain a 3D normal vector from a std::string line.
- vector< vector< int > > **ObjLoader::parseFaceElements** (string line)
  parses face elements into a vector of 3 vectors of 3 ints from an .obj line
- void **ObjLoader::parseElementTriple** (string triple, vector< int > &v_elements, vector< int > &uv_elements, vector< int > &n_elements)
  Helper function that splits an obj element listing (x/y/z) into their proper components (vertices, UVs, normals.)
- Object * **ObjLoader::loadObj** (Scene &scene, const char *filename, const char *defaultObjName)
  loadObj loads all available objects from a .obj file into the provided scene.
- void **ObjLoader::loadModelFromFile** (Object *object, const char *filename)
  Legacy function until I make everything suck less: Loads a single model from an OBJ and stores it into a single Object instance.

5.19.1 Detailed Description

Implementation for reading in geometry from .OBJ files.

Date

2013-03-14

Authors

Greg Giannone, Nick VerVoort, John Huston

Definition in file ObjLoader.cpp.

5.19.2 Macro Definition Documentation

5.19.2.1 #define RAND_FLOAT (rand() / (float) RAND_MAX)

RAND_FLOAT returns a random float from (0,1).

Definition at line 287 of file ObjLoader.cpp.

5.20 ObjLoader.hpp File Reference

Headers for functions for reading in geometry from .OBJ files.

```
#include <vector>
#include <string>
#include <exception>
#include <sstream>
#include "vec.hpp"
#include "Object.hpp"
```

Functions

- const vector< string > **ObjLoader::split** (const string &str, const char delim)

  Split a string by an arbitrary delimiter.
- Object * **ObjLoader::loadObj** (Scene &scene, const char *filename, const char *defaultObjName)

  loadObj loads all available objects from a .obj file into the provided scene.
- vec4 **ObjLoader::parseVertex** (string line)

  Obtain a vertex from a std::string line.
- vec2 **ObjLoader::parseTextureUV** (string line)

  Obtain a 2D TexUV coordinate from a std::string line.
- vec3 **ObjLoader::parseNormal** (string line)

  Obtain a 3D normal vector from a std::string line.
- vector< vector< int > > **ObjLoader::parseFaceElements** (string line)

  parses face elements into a vector of 3 vectors of 3 ints from an .obj line
- void **ObjLoader::parseElementTriple** (string triple, vector< int > &v_elements, vector< int > &uv_elements, vector< int > &n_elements)

  Helper function that splits an obj element listing (x/y/z) into their proper components (vertices, UVs, normals.)
- void **ObjLoader::loadModelFromFile** (Object *object, const char *filename)

  Legacy function until I make everything suck less: Loads a single model from an OBJ and stores it into a single Object instance.

5.20.1 Detailed Description

Headers for functions for reading in geometry from .OBJ files.

Date

2013-03-14

Authors

Greg Giannone, Nick VerVoort, John Huston

Definition in file ObjLoader.hpp.

5.21 terrain.cpp File Reference

This is a trimmed version of our Fall 2012 project.

```
#include <cmath>
#include <cstdio>
#include <sstream>
#include <cstdlib>
#include <time.h>
#include "platform.h"
#include "globals.h"
#include "vec.hpp"
#include "mat.hpp"
#include "Engine.hpp"
#include "Camera.hpp"
#include "Cameras.hpp"
#include "Screen.hpp"
#include "Object.hpp"
#include "Timer.hpp"
#include "Scene.hpp"
#include "model.hpp"
#include "InitShader.hpp"
#include "glut_callbacks.h"
#include "ObjLoader.hpp"
```

Typedefs

- typedef Angel::vec4 **color4**
- typedef Angel::vec4 **point4**

Functions

- void randomize_terrain ()

  randomize_terrain is called to regenerate the terrain in this application.
- void init ()

  init will initialize this particular flythrough, by creating and instantiating a shader, a camera, and a number of initial objects.
- void cleanup (void)

  cleanup is a routine to call at exit time that will free up the resources the application is using.
- void displayViewport (void)

  displayViewport is responsible for drawing a single viewport.
- void display (void)

  display is responsible for drawing an entire screen.
- void TerrainGenerationAnimation (TransCache &obj)

  TerrainGenerationAnimation is an animation callback that will: (A) If triggered, begin to shrink the object into a flat plane, (B) Order a re-generation of the terrain data (C) will grow the object back into its new, final shape.
- void wiilook (Camera &WiiCamera, const Angel::vec3 &NewTheta, const Angel::vec3 &MovementRates)

  wiilook is an analog of mouselook, for wii remote controls.
- void **simpleRotateY** (TransCache &obj)
- void **simpleRotateAnim** (TransCache &obj)
- void **animationTest** (TransCache &obj)
- void **idle** (void)
- void **menufunc** (int value)
- int **main** (int argc, char **argv)

Variables

- const char * **terrainTex** [ ]
- bool **switchingTerrain** = false
- float heightScale = 0.0

  hackity hack hack hackey doo!
- float **ticker** = 0.0

5.21.1 Detailed Description

This is a trimmed version of our Fall 2012 project.

Authors

John Huston, Nicholas StPierre, Chris Compton

Date

2013-02-23

This is a tech demo for terrain generation using an udpated engine derived from Ed Angel's code from his book.

Definition in file terrain.cpp.

5.21.2 Function Documentation

5.21.2.1 void cleanup (void)

cleanup is a routine to call at exit time that will free up the resources the application is using.

While not critical, it does aid in using debuggers to not have any memory leaks at exit time.

Returns

void.

Definition at line 230 of file terrain.cpp.

5.21.2.2 void display (void)

display is responsible for drawing an entire screen.

Returns

void.

Definition at line 255 of file terrain.cpp.

5.21.2.3 void displayViewport (void)

displayViewport is responsible for drawing a single viewport.

Returns

void.

Definition at line 241 of file terrain.cpp.

5.21.2.4 void init (void)

init will initialize this particular flythrough, by creating and instantiating a shader, a camera, and a number of initial objects.

Returns

void.

Definition at line 91 of file terrain.cpp.

5.21.2.5 void randomize_terrain ( )

randomize_terrain is called to regenerate the terrain in this application.

It assumes the terrain is named "terrain" and is located as a direct child of the main scene graph.

Returns

void.

Definition at line 67 of file terrain.cpp.

5.21.2.6 void TerrainGenerationAnimation (TransCache & *obj*)

TerrainGenerationAnimation is an animation callback that will: (A) If triggered, begin to shrink the object into a flat plane, (B) Order a re-generation of the terrain data (C) will grow the object back into its new, final shape.

Parameters

obj	A reference to an object's transformation state.

Returns

void.

Definition at line 279 of file terrain.cpp.

5.21.2.7 void wiilook (Camera & *WiiCamera,* const Angel::vec3 & *NewTheta,* const Angel::vec3 & *MovementRates*)

wiilook is an analog of mouselook, for wii remote controls.

It takes a reference to a Camera, and two vec3s, and uses the information to adjust the Camera's rotation.

Parameters

WiiCamera	The camera to adjust the rotation of.
NewTheta	The X,Y,Z angles of the Wii Remote.
MovementRates	The X, Y, Z angular velocities of the Wii Remote.

Returns

Void.

Definition at line 346 of file terrain.cpp.

5.21.3 Variable Documentation

5.21.3.1 const char* terrainTex[ ]

Initial value:

```
= {
  "../Textures/GoodTextures_0013423.jpg",
  "../Textures/GoodTextures_0013779.jpg",
  "../Textures/GrassGreenTexture0002.jpg",
  "../Textures/GoodTextures_0013418.jpg",
  "../Textures/GoodTextures_0013291.jpg"
}
```

Definition at line 52 of file terrain.cpp.

5.22 vec.cpp File Reference

Implementation for the vec2, vec3, and vec4 classes.

```
#include "vec.hpp"
#include <cmath>
#include "globals.h"
```

Functions

- vec2 **Angel::operator*** (const GLfloat s, const vec2 &v)
- std::ostream & **Angel::operator<<** (std::ostream &os, const vec2 &v)
- std::istream & **Angel::operator>>** (std::istream &is, vec2 &v)
- GLfloat **Angel::dot** (const vec2 &u, const vec2 &v)
- GLfloat **Angel::length** (const vec2 &v)
- vec2 **Angel::normalize** (const vec2 &v)
- vec3 **Angel::operator*** (const GLfloat s, const vec3 &v)
- std::ostream & **Angel::operator<<** (std::ostream &os, const vec3 &v)
- std::istream & **Angel::operator>>** (std::istream &is, vec3 &v)
- GLfloat **Angel::dot** (const vec3 &u, const vec3 &v)
- GLfloat **Angel::length** (const vec3 &v)
- vec3 **Angel::normalize** (const vec3 &v)
- vec3 **Angel::cross** (const vec3 &a, const vec3 &b)
- vec4 **Angel::operator*** (const GLfloat s, const vec4 &v)
- std::ostream & **Angel::operator<<** (std::ostream &os, const vec4 &v)
- std::istream & **Angel::operator>>** (std::istream &is, vec4 &v)
- GLfloat **Angel::dot** (const vec4 &u, const vec4 &v)
- GLfloat **Angel::length** (const vec4 &v)
- vec4 **Angel::normalize** (const vec4 &v)
- vec3 **Angel::cross** (const vec4 &a, const vec4 &b)
- vec3 **Angel::XYZ** (const vec4 &a)

5.22.1 Detailed Description

Implementation for the vec2, vec3, and vec4 classes.

Author

Ed Angel

Date

2012-12-04

Modified heavily from code available from Ed Angel's website, http://www.cs.unm.edu/~angel/BOOK/-INTERACTIVE_COMPUTER_GRAPHICS/SIXTH_EDITION/ Published from his book, Interactive Computer Graphics A Top-Down Approach with OpenGL, Sixth Edition Addison-Wesley 2012

Definition in file vec.cpp.

Index