RayTracer Starter Code Reference Manual

Generated by Doxygen 1.5.1-p1

Wed Apr 11 20:04:07 2007

Contents

1 RayTracer Starter Code Namespace Index

1.1 RayTracer Starter Code Namespace List

Here is a list of all namespaces with brief descriptions:

2 RayTracer Starter Code Class Index

2.1 RayTracer Starter Code Class List

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

3 RayTracer Starter Code File Index

3.1 RayTracer Starter Code File List

Here is a list of all files with brief descriptions:

4 RayTracer Starter Code Namespace Documentation

4.1 std Namespace Reference

5 RayTracer Starter Code Class Documentation

5.1 rc_Spline Class Reference

class to represent a spline

`#include <rc_spline.h>`

Public Member Functions

- void **addPoint** (const **Vec3f** &v)
  add a point to the spline segment
- int **length** ()
  get the length of the spline segment
- **pointVector** & **points** ()
  accessor to the vector of control points
- void **loadSplineFrom** (char *filename)
  load the definition of this spline from a file

Private Member Functions

- void **loadSegmentFrom** (char *filename)
  load the definition of this spline segment from a file

Private Attributes

- **pointVector m_vPoints**
  vector of control points

5.1.1 Detailed Description

class to represent a spline

5.1.2 Member Function Documentation

5.1.2.1 void rc_Spline::loadSegmentFrom (char * *filename*) [private]

load the definition of this spline segment from a file

Parameters:

filename file containing the definition for this spline segment

5.1.2.2 void rc_Spline::addPoint (const Vec3f & *v*) [inline]

add a point to the spline segment

Parameters:

v point to add

5.1.2.3 int rc_Spline::length () [inline]

get the length of the spline segment

5.1.2.4 pointVector& rc_Spline::points () [inline]

accessor to the vector of control points

Normaly used for iterating, e.g. for(pointVectorIter iter = pSpline->start(); iter != pSpline->end(); iter++) { **Vec3f** (p. 12) curpt = *iter; // do whatever is needed with each point }

5.1.2.5 void rc_Spline::loadSplineFrom (char * *filename*)

load the definition of this spline from a file

Parameters:

filename file containing the definition for this spline

5.1.3 Member Data Documentation

5.1.3.1 pointVector rc_Spline::m_vPoints [private]

vector of control points

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

- **rc_spline.h**
- **rc_spline.cpp**

5.2 Vec2f Class Reference

`#include <util_vectors.h>`

Public Member Functions

- **Vec2f** ()

  Null Constructor for class ***Vec2f*** *(*p. 4*).*

- **Vec2f** (const **Vec2f** &V)

  Copy Constructor for class ***Vec2f*** *(*p. 4*).*

- **Vec2f** (float d0, float d1)

  Constructor for class ***Vec2f*** *(*p. 4*) from two floats representing the coordinates.*

- **Vec2f** (const **Vec2f** &V1, const **Vec2f** &V2)

  Constructor for class ***Vec2f*** *(*p. 4*) as a difference from two vectors (points).*

- ~**Vec2f** ()

  Destructor.

- void **Get** (float &d0, float &d1) const

  Accessor to both fields simultaneously.

- float **operator[]** (int i) const

  Array like accessor to vector fields (overloading the array operator []).

- float **x** () const

  Standard accessor for the first coordinate.

- float & **x** ()

  Standard accessor for the first coordinate writable version.

- float **y** () const

  Standard accessor for the second coordinate.

- float & **y** ()

  Standard accessor for the first coordinate writable version.

- float **Length** () const

  Compute the length (norm) of the vector.

- void **Set** (float d0, float d1)

  Accessor to set both fields simultaneously.

- void **Scale** (float d0, float d1)

  Non uniform scaling of the vector.

- void **Divide** (float d0, float d1)

  Non uniform inverse scaling of the vector.

- void **Negate** ()

  Negate the vector (flip the signs of each component).

- **Vec2f** & **operator**= (const **Vec2f** &V)

  *Copy operator for class **Vec2f** (p. 4) (overloading =).*

- int **operator**== (const **Vec2f** &V) const

  *Equality comparison operator for class **Vec2f** (p. 4) (overloading ==).*

- int **operator!**= (const **Vec2f** &V)

  *Inequality comparison operator for class **Vec2f** (p. 4) (overloading !=).*

- **Vec2f** & **operator**+= (const **Vec2f** &V)

  Vector addition (overloading +=).

- **Vec2f** & **operator-**= (const **Vec2f** &V)
  Vector subtraction (overloading -=).
- **Vec2f** & **operator** ∗= (float f)
  Vector scalar multiplication (overloading ∗=).
- **Vec2f** & **operator**/= (float f)
  Vector scalar division (overloading /=).
- float **Dot2** (const **Vec2f** &V) const
  Dot (scalar) product.
- void **Write** (FILE ∗F=stdout)
  Write the vector to a file.

Static Public Member Functions

- static void **Add** (**Vec2f** &a, const **Vec2f** &b, const **Vec2f** &c)
  Addition of two vectors.
- static void **Sub** (**Vec2f** &a, const **Vec2f** &b, const **Vec2f** &c)
  Subtraction of two vectors.
- static void **CopyScale** (**Vec2f** &a, const **Vec2f** &b, float c)
  Copy a scaled version of b to a (a =c∗b).
- static void **AddScale** (**Vec2f** &a, const **Vec2f** &b, const **Vec2f** &c, float d)
  Add a scaled version of c to b (a = b+c∗d).
- static void **Average** (**Vec2f** &a, const **Vec2f** &b, const **Vec2f** &c)
  Average two vectors.
- static void **WeightedSum** (**Vec2f** &a, const **Vec2f** &b, float c, const **Vec2f** &d, float e)
  Compute the weighted sum of two vectors.

Private Attributes

- float **data** [2]
  Data members.

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Vec2f::Vec2f () [inline]

Null Constructor for class **Vec2f** (p. 4).

5.2.1.2 Vec2f::Vec2f (const Vec2f & *V*) [inline]

Copy Constructor for class **Vec2f** (p. 4).

Parameters:

V Vector to copy from

5.2.1.3 Vec2f::Vec2f (float *d0*, float *d1*) [inline]

Constructor for class **Vec2f** (p. 4) from two floats representing the coordinates.

Parameters:

d0 First vector coordinate

d1 Second vector coordinate

5.2.1.4 Vec2f::Vec2f (const Vec2f & *V1*, const Vec2f & *V2*) [inline]

Constructor for class **Vec2f** (p. 4) as a difference from two vectors (points).

You can use this constructor to build a vector from two points.

Parameters:

V1 First vector

V2 Second vector

5.2.1.5 Vec2f::~Vec2f () [inline]

Destructor.

5.2.2 Member Function Documentation

5.2.2.1 void Vec2f::Get (float & *d0*, float & *d1*) const [inline]

Accessor to both fields simultaneously.

Load the two vector coordinates to variables d0 and d1.

Parameters:

d0 First vector coordinate

d1 Second vector coordinate

5.2.2.2 float Vec2f::operator[] (int *i*) const [inline]

Array like accessor to vector fields (overloading the array operator []).

Access the fields of a vector as if it where an array, e.g. first coordinate: vector[0]

Parameters:

i component to be accessed

5.2.2.3 float Vec2f::x () const [inline]

Standard accessor for the first coordinate.

5.2.2.4 float& Vec2f::x () [inline]

Standard accessor for the first coordinate writable version.

5.2.2.5 float Vec2f::y () const [inline]

Standard accessor for the second coordinate.

5.2.2.6 float& Vec2f::y () [inline]

Standard accessor for the first coordinate writable version.

5.2.2.7 float Vec2f::Length () const [inline]

Compute the length (norm) of the vector.

5.2.2.8 void Vec2f::Set (float *d0*, float *d1*) [inline]

Accessor to set both fields simultaneously.

Load the vector coordinates from variables d0 and d1.

Parameters:

d0 First vector coordinate

d1 Second vector coordinate

5.2.2.9 void Vec2f::Scale (float *d0*, float *d1*) [inline]

Non uniform scaling of the vector.

Parameters:

d0 scaling of the first vector coordinate
d1 scaling of the second vector coordinate

5.2.2.10 void Vec2f::Divide (float *d0*, float *d1*) [inline]

Non uniform inverse scaling of the vector.

Parameters:

d0 inverse scaling of the first vector coordinate
d1 inverse scaling of the second vector coordinate

5.2.2.11 void Vec2f::Negate () [inline]

Negate the vector (flip the signs of each component).

5.2.2.12 Vec2f& Vec2f::operator= (const Vec2f & *V*) [inline]

Copy operator for class **Vec2f** (p. 4) (overloading =).

Parameters:

V Vector to copy from

5.2.2.13 int Vec2f::operator== (const Vec2f & *V*) const [inline]

Equality comparison operator for class **Vec2f** (p. 4) (overloading ==).

Parameters:

V Vector to compare to

5.2.2.14 int Vec2f::operator!= (const Vec2f & *V*) [inline]

Inequality comparison operator for class **Vec2f** (p. 4) (overloading !=).

Parameters:

V Vector to compare to

5.2.2.15 Vec2f& Vec2f::operator+= (const Vec2f & *V*) [inline]

Vector addition (overloading +=).

Parameters:

V Vector (point) to add

5.2.2.16 Vec2f& Vec2f::operator-= (const Vec2f & *V*) [inline]

Vector subtraction (overloading -=).

Parameters:

V Vector (point) to add

5.2.2.17 Vec2f& Vec2f::operator *= (float *f*) [inline]

Vector scalar multiplication (overloading *=).

Parameters:

f scalar to multiply vector by

5.2.2.18 Vec2f& Vec2f::operator/= (float *f*) [inline]

Vector scalar division (overloading /=).

Parameters:

f scalar to divide vector by

5.2.2.19 float Vec2f::Dot2 (const Vec2f & *V*) const [inline]

Dot (scalar) product.

5.2.2.20 static void Vec2f::Add (Vec2f & *a*, const Vec2f & *b*, const Vec2f & *c*) [inline, static]

Addition of two vectors.

Parameters:

a Vector to store the results of the addition of b+c

b First addition operand

c Second addition operand

5.2.2.21 static void Vec2f::Sub (Vec2f & *a*, const Vec2f & *b*, const Vec2f & *c*) [inline, static]

Subtraction of two vectors.

Parameters:

- ***a*** Vector to store the results of the subtraction of b0-c
- ***b*** First subtraction operand
- ***c*** Second subtraction operand

5.2.2.22 static void Vec2f::CopyScale (Vec2f & *a*, const Vec2f & *b*, float *c*) [inline, static]

Copy a scaled version of b to a (a =c∗b).

Parameters:

- ***a*** Vector to store the results of the subtraction of c∗b
- ***b*** Vector operand
- ***c*** Scalar operand

5.2.2.23 static void Vec2f::AddScale (Vec2f & *a*, const Vec2f & *b*, const Vec2f & *c*, float *d*) [inline, static]

Add a scaled version of c to b (a = b+c∗d).

Parameters:

- ***a*** Vector to store the results of the subtraction of a= b+c∗d
- ***b*** First vector operand
- ***c*** Second vector operand
- ***d*** Scalar operand

5.2.2.24 static void Vec2f::Average (Vec2f & *a*, const Vec2f & *b*, const Vec2f & *c*) [inline, static]

Average two vectors.

Parameters:

- ***a*** Vector to store the results of (b+c/2)
- ***b*** First vector operand
- ***c*** Second vector operand

5.2.2.25 static void Vec2f::WeightedSum (Vec2f & *a*, const Vec2f & *b*, float *c*, const Vec2f & *d*, float *e*) [inline, static]

Compute the weighted sum of two vectors.

Parameters:

- ***a*** Vector to store the results of (b∗c+d∗e)
- ***b*** First vector operand
- ***c*** First scalar operand
- ***d*** Second vector operan
- ***e*** Second scalar operand

5.2.2.26 void Vec2f::Write (FILE ∗ *F* = stdout) [inline]

Write the vector to a file.

Parameters:

- ***F*** pointer to a FILE structure (standard output if ommited)

5.2.3 Member Data Documentation

5.2.3.1 float Vec2f::data[2] [private]

Data members.

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

- **util_vectors.h**

5.3 Vec3f Class Reference

`#include <util_vectors.h>`

Public Member Functions

- **Vec3f** ()
  Null Constructor for class ***Vec2f*** *(*p. 4*).*
- **Vec3f** (const **Vec3f** &V)
  Copy Constructor for class ***Vec2f*** *(*p. 4*).*
- **Vec3f** (float d0, float d1, float d2)

*Constructor for class **Vec2f** (p. 4) from three floats representing the coordinates.*

- **Vec3f** (const **Vec3f** &V1, const **Vec3f** &V2)

  *Constructor for class **Vec2f** (p. 4) as a difference from two vectors (points).*

- ~**Vec3f** ()

  Destructor.

- void **Get** (float &d0, float &d1, float &d2) const

  Accessor to the three data members simultaneously.

- float **operator[]** (int i) const

  Array like accessor to vector fields (overloading the array operator []).

- float **x** () const

  Standard accessor for the first coordinate.

- float & **x** ()

  Standard accessor for the first coordinate writeable version.

- float **y** () const

  Standard accessor for the second coordinate.

- float & **y** ()

  Standard accessor for the second coordinate writeable version.

- float **z** () const

  Standard accessor for the third coordinate.

- float & **z** ()

  Standard accessor for the third coordinate writeable version.

- float **r** () const

  Alternative accessor for the first coordinate (red).

- float **g** () const

  Alternative accessor for the second coordinate (green).

- float **b** () const

  Alternative accessor for the third coordinate (blue).

- float **Length** () const

Compute the length (norm) of the vector.

- void **Set** (float d0, float d1, float d2)

  Accessor to set the three fields simultaneously.

- void **Scale** (float d0, float d1, float d2)

  Non uniform scaling of the vector.

- void **Divide** (float d0, float d1, float d2)

  Non uniform inverse scaling of the vector.

- void **Normalize** ()

  Normalize a vector to unit length.

- void **Negate** ()

  Negate the vector (flip the signs of each component).

- **Vec3f** & **operator**= (const **Vec3f** &V)

  *Copy operator for class **Vec2f** (p. 4) (overloading =).*

- int **operator**== (const **Vec3f** &V)

  *Equality comparison operator for class **Vec2f** (p. 4) (overloading ==).*

- int **operator!**= (const **Vec3f** &V)

  *Inequality comparison operator for class **Vec2f** (p. 4) (overloading !=).*

- **Vec3f** & **operator**+= (const **Vec3f** &V)

  Vector addition (overloading +=).

- **Vec3f** & **operator-**= (const **Vec3f** &V)

  Vector subtraction (overloading -=).

- **Vec3f** & **operator** ∗= (int i)

  Vector integer scalar multiplication (overloading ∗=).

- **Vec3f** & **operator** ∗= (float f)

  Vector float scalar multiplication (overloading ∗=).

- **Vec3f** & **operator**/= (int i)

  Vector integer scalar division (overloading /=).

- **Vec3f** & **operator**/= (float f)

Vector float scalar division (overloading /=).

- float **Dot3** (const **Vec3f** &V) const
  Dot (scalar) product.
- void **Write** (FILE *F=stdout)
  Write the vector to a file.

Static Public Member Functions

- static void **Cross3** (**Vec3f** &c, const **Vec3f** &v1, const **Vec3f** &v2)
  Compute the cross product of two vectors.

Private Attributes

- float **data** [3]
  Data members.

Friends

- **Vec3f operator**+ (const **Vec3f** &v1, const **Vec3f** &v2)
  Addition of two vectors (overloading +).
- **Vec3f operator-** (const **Vec3f** &v1, const **Vec3f** &v2)
  Subtraction of two vectors (overloading -).
- **Vec3f operator** * (const **Vec3f** &v1, float f)
  Scalar vector multiplication.
- void **Add** (**Vec3f** &a, const **Vec3f** &b, const **Vec3f** &c)
  Addition of two vectors.
- void **Sub** (**Vec3f** &a, const **Vec3f** &b, const **Vec3f** &c)
  Subtraction of two vectors.
- void **CopyScale** (**Vec3f** &a, const **Vec3f** &b, float c)
  *Copy a scaled version of b to a (a =c*b).*
- void **AddScale** (**Vec3f** &a, const **Vec3f** &b, const **Vec3f** &c, float d)

*Add a scaled version of c to b (a = b+c*d).*

- void **Average** (**Vec3f** &a, const **Vec3f** &b, const **Vec3f** &c)
  Average two vectors.
- void **WeightedSum** (**Vec3f** &a, const **Vec3f** &b, float c, const **Vec3f** &d, float e)
  Compute the weighted sum of two vectors.

5.3.1 Constructor & Destructor Documentation

5.3.1.1 Vec3f::Vec3f () [inline]

Null Constructor for class **Vec2f** (p. 4).

5.3.1.2 Vec3f::Vec3f (const Vec3f & *V*) [inline]

Copy Constructor for class **Vec2f** (p. 4).

Parameters:

V Vector to copy from

5.3.1.3 Vec3f::Vec3f (float *d0*, float *d1*, float *d2*) [inline]

Constructor for class **Vec2f** (p. 4) from three floats representing the coordinates.

Parameters:

d0 First vector coordinate
d1 Second vector coordinate
d2 Thrid vector coordinate

5.3.1.4 Vec3f::Vec3f (const Vec3f & *V1*, const Vec3f & *V2*) [inline]

Constructor for class **Vec2f** (p. 4) as a difference from two vectors (points).

You can use this constructor to build a vector from two points.

Parameters:

V1 First vector
V2 Second vector

5.3.1.5 Vec3f::~Vec3f () [inline]

Destructor.

5.3.2 Member Function Documentation

5.3.2.1 void Vec3f::Get (float & *d0*, float & *d1*, float & *d2*) const [inline]

Accessor to the three data members simultaneously.

Load the two vector coordinates to variables d0, d1 and d2.

Parameters:

d0 First vector coordinate

d1 Second vector coordinate

d2 Second vector coordinate

5.3.2.2 float Vec3f::operator[] (int *i*) const [inline]

Array like accessor to vector fields (overloading the array operator []).

Access the fields of a vector as if it where an array, e.g. first coordinate: vector[0]

Parameters:

i component to be accessed

5.3.2.3 float Vec3f::x () const [inline]

Standard accessor for the first coordinate.

5.3.2.4 float& Vec3f::x () [inline]

Standard accessor for the first coordinate writeable version.

5.3.2.5 float Vec3f::y () const [inline]

Standard accessor for the second coordinate.

5.3.2.6 float& Vec3f::y () [inline]

Standard accessor for the second coordinate writeable version.

5.3.2.7 float Vec3f::z () const [inline]

Standard accessor for the third coordinate.

5.3.2.8 float& Vec3f::z () [inline]

Standard accessor for the third coordinate writeable version.

5.3.2.9 float Vec3f::r () const [inline]

Alternative accessor for the first coordinate (red).

Useful if you use **Vec3f** (p. 12) to represent colors.

5.3.2.10 float Vec3f::g () const [inline]

Alternative accessor for the second coordinate (green).

Useful if you use **Vec3f** (p. 12) to represent colors.

5.3.2.11 float Vec3f::b () const [inline]

Alternative accessor for the third coordinate (blue).

Useful if you use **Vec3f** (p. 12) to represent colors.

5.3.2.12 float Vec3f::Length () const [inline]

Compute the length (norm) of the vector.

5.3.2.13 void Vec3f::Set (float *d0*, float *d1*, float *d2*) [inline]

Accessor to set the three fields simultaneously.

Load the vector coordinates from variables d0,d1 and d2.

Parameters:

d0 First vector coordinate

d1 Second vector coordinate

d2 Third vector coordinate

5.3.2.14 void Vec3f::Scale (float *d0*, float *d1*, float *d2*) [inline]

Non uniform scaling of the vector.

Parameters:

d0 scaling of the first vector coordinate

d1 scaling of the second vector coordinate

d2 scaling of the third vector coordinate

5.3.2.15 void Vec3f::Divide (float *d0*, float *d1*, float *d2*) [inline]

Non uniform inverse scaling of the vector.

Parameters:

d0 inverse scaling of the first vector coordinate

d1 inverse scaling of the second vector coordinate

d2 scaling of the third vector coordinate

5.3.2.16 void Vec3f::Normalize () [inline]

Normalize a vector to unit length.

5.3.2.17 void Vec3f::Negate () [inline]

Negate the vector (flip the signs of each component).

5.3.2.18 Vec3f& Vec3f::operator= (const Vec3f & *V*) [inline]

Copy operator for class **Vec2f** (p. 4) (overloading =).

Parameters:

V Vector to copy from

5.3.2.19 int Vec3f::operator== (const Vec3f & *V*) [inline]

Equality comparison operator for class **Vec2f** (p. 4) (overloading ==).

Parameters:

V Vector to compare to

5.3.2.20 int Vec3f::operator!= (const Vec3f & V) [inline]

Inequality comparison operator for class **Vec2f** (p. 4) (overloading !=).

Parameters:

V Vector to compare to

5.3.2.21 Vec3f& Vec3f::operator+= (const Vec3f & V) [inline]

Vector addition (overloading +=).

Parameters:

V Vector (point) to add

5.3.2.22 Vec3f& Vec3f::operator-= (const Vec3f & V) [inline]

Vector subtraction (overloading -=).

Parameters:

V Vector (point) to add

5.3.2.23 Vec3f& Vec3f::operator *= (int i) [inline]

Vector integer scalar multiplication (overloading *=).

Parameters:

i scalar to multiply vector by

5.3.2.24 Vec3f& Vec3f::operator *= (float f) [inline]

Vector float scalar multiplication (overloading *=).

Parameters:

f scalar to multiply vector by

5.3.2.25 Vec3f& Vec3f::operator/= (int i) [inline]

Vector integer scalar division (overloading /=).

Parameters:

i integer scalar to divide vector by

5.3.2.26 Vec3f& Vec3f::operator/= (float *f*) [inline]

Vector float scalar division (overloading /=).

Parameters:

f float scalar to divide vector by

5.3.2.27 float Vec3f::Dot3 (const Vec3f & *V*) const [inline]

Dot (scalar) product.

5.3.2.28 static void Vec3f::Cross3 (Vec3f & *c*, const Vec3f & *v1*, const Vec3f & *v2*) [inline, static]

Compute the cross product of two vectors.

Parameters:

c Vector to store the results of (v1 x v2)

v1 First vector operand

v2 Second vector operan

5.3.2.29 void Vec3f::Write (FILE * *F* = stdout) [inline]

Write the vector to a file.

Parameters:

F pointer to a FILE structure (standard output if ommited)

5.3.3 Friends And Related Function Documentation

5.3.3.1 Vec3f operator+ (const Vec3f & *v1*, const Vec3f & *v2*) [friend]

Addition of two vectors (overloading +).

Parameters:

v1 First addition operand

v2 Second addition operand

5.3.3.2 Vec3f operator- (const Vec3f & *v1*, const Vec3f & *v2*) [friend]

Subtraction of two vectors (overloading -).

Parameters:

v1 First addition operand

v2 Second addition operand

5.3.3.3 Vec3f operator * (const Vec3f & *v1*, float *f*) [friend]

Scalar vector multiplication.

Parameters:

v1 Vector operand

f Scalar operand

5.3.3.4 void Add (Vec3f & *a*, const Vec3f & *b*, const Vec3f & *c*) [friend]

Addition of two vectors.

Parameters:

a Vector to store the results of the addition of b+c

b First addition operand

c Second addition operand

5.3.3.5 void Sub (Vec3f & *a*, const Vec3f & *b*, const Vec3f & *c*) [friend]

Subtraction of two vectors.

Parameters:

a Vector to store the results of the subtraction of b0-c

b First subtraction operand

c Second subtraction operand

5.3.3.6 void CopyScale (Vec3f & *a*, const Vec3f & *b*, float *c*) [friend]

Copy a scaled version of b to a (a =c∗b).

Parameters:

- ***a*** Vector to store the results of the subtraction of c∗b
- ***b*** Vector operand
- ***c*** Scalar operand

5.3.3.7 void AddScale (Vec3f & *a*, const Vec3f & *b*, const Vec3f & *c*, float *d*) [friend]

Add a scaled version of c to b (a = b+c∗d).

Parameters:

- ***a*** Vector to store the results of the subtraction of a= b+c∗d
- ***b*** First vector operand
- ***c*** Second vector operand
- ***d*** Scalar operand

5.3.3.8 void Average (Vec3f & *a*, const Vec3f & *b*, const Vec3f & *c*) [friend]

Average two vectors.

Parameters:

- ***a*** Vector to store the results of (b+c/2)
- ***b*** First vector operand
- ***c*** Second vector operand

5.3.3.9 void WeightedSum (Vec3f & *a*, const Vec3f & *b*, float *c*, const Vec3f & *d*, float *e*) [friend]

Compute the weighted sum of two vectors.

Parameters:

- ***a*** Vector to store the results of (b∗c+d∗e)
- ***b*** First vector operand
- ***c*** First scalar operand
- ***d*** Second vector operan
- ***e*** Second scalar operand

5.3.4 Member Data Documentation

5.3.4.1 float Vec3f::data[3] [private]

Data members.

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

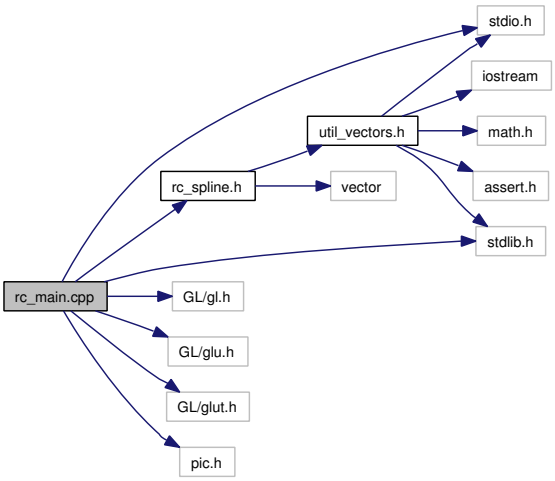
- **util_vectors.h**

6 RayTracer Starter Code File Documentation

6.1 rc_main.cpp File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <pic.h>
#include "rc_spline.h"
```

Include dependency graph for rc_main.cpp:



Defines

- #define **WINDOW_WIDTH** 640
- #define **WINDOW_HEIGHT** 480

Functions

- void **InitGL** (GLvoid)
- void **doIdle** ()
- void **display** (void)
- void **keyboardfunc** (unsigned char key, int x, int y)
- void **menufunc** (int value)
- int **main** (int argc, char **argv)

Variables

- **rc_Spline g_Track**
- int **g_iMenuId**

6.1.1 Define Documentation

6.1.1.1 #define WINDOW_HEIGHT 480

6.1.1.2 #define WINDOW_WIDTH 640

6.1.2 Function Documentation

6.1.2.1 void display (void)

6.1.2.2 void doIdle ()

6.1.2.3 void InitGL (GLvoid)

6.1.2.4 void keyboardfunc (unsigned char *key*, int *x*, int *y*)

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

6.1.2.6 void menufunc (int *value*)

6.1.3 Variable Documentation

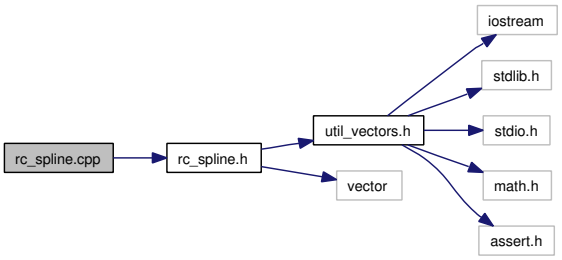
6.1.3.1 int g_iMenuId

6.1.3.2 rc_Spline g_Track

6.2 rc_spline.cpp File Reference

```
#include "rc_spline.h"
```

Include dependency graph for rc_spline.cpp:

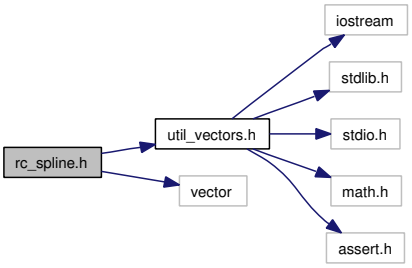


6.3 rc_spline.h File Reference

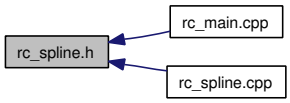
contains the classes to represent and load splines

```
#include "util_vectors.h"
#include <vector>
```

Include dependency graph for rc_spline.h:



This graph shows which files directly or indirectly include this file:



Classes

- class **rc_Spline**
  class to represent a spline

Typedefs

- typedef std::vector< **Vec3f** > **pointVector**
  type alias for a vector of ***Vec3f*** *(p. 12)*
- typedef pointVector::iterator **pointVectorIter**
  type alias for an iterator of a pointVector

6.3.1 Detailed Description

contains the classes to represent and load splines

Author:

Created by Roberto Lublinerman on Mon Feb 20 2007.

CSE 418: Programming Project 4

6.3.2 Typedef Documentation

6.3.2.1 typedef std::vector<Vec3f> pointVector

type alias for a vector of **Vec3f** (p. 12)

6.3.2.2 typedef pointVector::iterator pointVectorIter

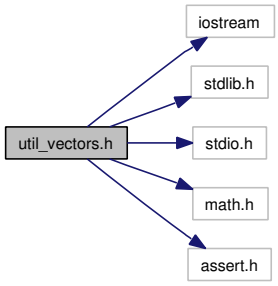
type alias for an iterator of a pointVector

6.4 util_vectors.h File Reference

contains 2- and 3-vector classes

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <assert.h>
```

Include dependency graph for util_vectors.h:



This graph shows which files directly or indirectly include this file:

Namespaces

- namespace **std**

Classes

- class **Vec2f**
- class **Vec3f**

Typedefs

- typedef **Vec3f Color3**

*Colors can be represented by **Vec3f** (p. 12): Color3.*

Functions

- ostream & **operator**<< (ostream &os, const **Vec3f** &v)
  Ouput the vector to a stream.

6.4.1 Detailed Description

contains 2- and 3-vector classes

Author:

Created by Sriram Vaidhyanathan on Thu Feb 26 2004.
Modified byRoberto Lublinerman on Mon Feb 19 2007.

CSE 418: Programming Project 3

6.4.2 Typedef Documentation

6.4.2.1 typedef Vec3f Color3

Colors can be represented by **Vec3f** (p. 12): Color3.

6.4.3 Function Documentation

6.4.3.1 ostream& operator<< (ostream & *os*, const Vec3f & *v*) [inline]

Ouput the vector to a stream.

Useful for debugging purposes

Index