Tartalomjegyzék

1. fejezet

Documentation of the enums

2. fejezet

Documentation of the interfaces

3. fejezet

Documentation of the classes

3.1. Class: AABB

3.1.1. Fields

	Name and Description
Protected	cornerFar
Protected	cornerNear
Protected	extents

3.1.2. Properties

3.1.3. Methods

	Name and Description
Public	AABB(const corner , const extents)
Public Virtual Constant	containsPoint(const pointToCheck , const Toperators)
Public Virtual Constant	fitsPoint(const pointToCheck , const Toperators)
Public Constant	getCorner ()
Public Constant	getExtents ()
Public Virtual Constant	intersect(const ray , const bounds , const T & scale , const Toperators)
Public	operator[](unsigned int index)
Public Constant	operator[](unsigned int index)
Public	setCorner(const corner)

Public AABB(const corner , const extents)

c'tor with arguments
 	
`corner:`
 	
- the point of the corner in the space, from which the extents will be measured
 	
`extents:`
 	
- the extents of the AABB for each dimension

Public Virtual Constant containsPoint(const pointToCheck , const Toperators)

This function can decide if a point is in the bounding volume, or not. (points of the surface are also true)

`point:`

- the point to be checked

`Toperators:`

- the operators object of type T

`Return:`

- true if the point is in the Bounding Volume (and false otherwise)

Public Virtual Constant fitsPoint(const pointToCheck , const Toperators)

This function can decide if a point fits the AABB or not.

`point:`

- the point to be checked

`Toperators:`

- the operators object of type T

`Return:`

- true if the point fits the AABB (and false otherwise)

Public Constant getCorner ()

This method can return the corner of the AABB .

`Return:`

- the corner of the AABB

Public Constant getExtents ()

This method can return the extents of the AABB for each dimension.

`Return:`

- the extents of the AABB

Public Virtual Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector . (0D Vector)
 	
`ray:`
 	
- the intersection ray
 	
`bounds:`
 	
- the bounds between the intersection will be looked for
 	
`scale:`
 	
- scale used by ray casting algorithms
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- The collection of intersections in ascending order of dimension time.

Public operator[](unsigned int index)
index operators They return the extent for the selected dimension.

Public setCorner(const corner)
This method can set the corner of the AABB . `corner:` - the new corner of the AABB

3.2. Class: BccGrid

3.2.1. Fields

	Name and Description
Protected	baseCc
Protected	translatedCc

3.2.2. Properties

3.2.3. Methods

	Name and Description
Public	BccGrid(const baseOrigin , const T & sideLength)
Public Virtual Constant	getFirstNeighbourCellVertices(const point)
Public Virtual Constant	getLerpVertices(const point , const Toperators)
Public Virtual Constant	getNearestPoint(const point , const Toperators)
Public Constant	getOrigin ()
Public Constant	getSideLength ()
Public	setOrigin(const baseOrigin)
Public	setSideLength(const T & sideLength)

Public BccGrid(const baseOrigin , const T & sideLength)

c'tor with arguments

`baseOrigin:`

- the origin point in world coordinates of the base CC

`sideLength:`

- the side length of the unit cell (should be a positive number)

Public Virtual Constant getFirstNeighbourCellVertices(const point)

This method gets a point, and returns in a vector all of the first neighbour points of this. The method doesn't check, if the point in the argument fits a lattice point, or not!

`point:`

- the point whose first neighbours we are looking for

`Return:`

- a vector of the first neighbours (each is given in world coordinates!)

Public Virtual Constant getLerpVertices(const point , const Toperators)

This method gets a point, and returns in a vector all of the vertices whose value influences the linear interpolation of the selected point.

`point:`

- the point whose lerp-vertices we are looking for

Public Virtual Constant getNearestPoint(const point , const Toperators)

This method can return the lattice point in whose Voronoi-region is the given point. This lattice point is the nearest neighbour point of the given point.

`point:`

- the point whose nearest neighbour we are looking for

`Toperators:`

- the operators object of type T

`Return:`

- the nearest neighbour point in the lattice in world coordinates (not in lattice locale coordinates!)

Public Constant getOrigin ()

This method returns the actual value of the origin of the grid.

`Return:`

- the actual origin of the grid

Public Constant getSideLength ()

This method returns the actual value of the side length of the unit cell.

`Return:`

- the actual side length of the unit cell

Public setOrigin(const baseOrigin)
This method can set origin of the grid. `origin:` - the new origin

Public setSideLength(const T & sideLength)
This method can set the side length (resolution) of the grid. `sideLength:` - the side length of the unit cell (should be a positive number)

3.3. Class: BoundingVolume

3.3.1. Fields

3.3.2. Properties

3.3.3. Methods

	Name and Description
Public Abstract Constant	containsPoint(const pointToCheck , const Toperators)
Public Abstract Constant	fitsPoint(const pointToCheck , const Toperators)
Public Abstract Constant	intersect(const ray , const bounds , const T & scale , const Toperators)

Public Abstract Constant containsPoint(const pointToCheck , const Toperators)

This function can decide if a point is in the bounding volume, or not. (points of the surface are also true)
 	
`point:`
 	
- the point to be checked
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- true if the point is in the Bounding Volume (and false otherwise)

Public Abstract Constant fitsPoint(const pointToCheck , const Toperators)

This function can decide if a point fits the surface of the Bounding Volume or not.
 	
`point:`
 	
- the point to be checked
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- true if the point fits the surface of the Bounding Volume (and false otherwise)

Public Abstract Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector . (0D Vector)
 	
`ray:`
 	
- the intersection ray
 	
`bounds:`
 	
- the bounds between the intersection will be looked for
 	
`scale:`
 	
- scale used by ray casting algorithms
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- The collection of intersections in ascending order of dimension time.

3.4. Class: CcGrid

3.4.1. Fields

	Name and Description
Protected	origin
Protected	sideLength
Protected	sideLengthHalf

3.4.2. Properties

3.4.3. Methods

	Name and Description
Public	CcGrid(const origin , const T & sideLength)
Public Virtual Constant	getFirstNeighbourCellVertices(const point)
Public Virtual Constant	getLerpVertices(const point , const Toperators)
Public Virtual Constant	getNearestPoint(const point , const Toperators)
Public Constant	getOrigin ()
Public Constant	getSideLength ()
Public	setOrigin(const origin)
Public	setSideLength(const T & sideLength)

Public CcGrid(const origin , const T & sideLength)

c'tor with arguments
 	
`origin:`
 	
- the origin point in world coordinates
 	
`sideLength:`
 	
- the side length of the unit cell (should be a positive number)

Public Virtual Constant getFirstNeighbourCellVertices(const point)

This method gets a point, and returns in a vector all of the first neighbour points of this. The method doesn't check, if the point in the argument fits a lattice point, or not!

`point:`

- the point whose first neighbours we are looking for

`Return:`

- a vector of the first neighbours (each is given in world coordinates!)

Public Virtual Constant getLerpVertices(const point , const Toperators)

This method gets a point, and returns in a vector all of the vertices whose value influences the linear interpolation of the selected point.

`point:`

- the point whose lerp-vertices we are looking for

Public Virtual Constant getNearestPoint(const point , const Toperators)

This method can return the lattice point in whose Voronoi-region is the given point. This lattice point is the nearest neighbour point of the given point.

`point:`

- the point whose nearest neighbour we are looking for

`Toperators:`

- the operators object of type T

`Return:`

- the nearest neighbour point in the lattice in world coordinates (not in lattice locale coordinates!)

Public Constant getOrigin ()

This method returns the actual value of the origin of the grid.

`Return:`

- the actual origin of the grid

Public Constant getSideLength ()

This method returns the actual value of the side length of the unit cell.

`Return:`

- the actual side length of the unit cell

Public setOrigin(const origin)

This method can set origin of the grid.

`origin:`

- the new origin

Public setSideLength(const T & sideLength)

This method can set the side length (resolution) of the grid.

`sideLength:`

- the side length of the unit cell (should be a positive number)

3.5. Class: Grid

3.5.1. Fields

3.5.2. Properties

3.5.3. Methods

	Name and Description
Public Abstract Constant	getFirstNeighbourCellVertices(const point)
Public Abstract Constant	getLerpVertices(const point , const Toperators)
Public Abstract Constant	getNearestPoint(const point , const Toperators)

Public Abstract Constant getFirstNeighbourCellVertices(const point)

This method gets a point, and returns in a vector all of the first neighbour points of this. The method doesn't check, if the point in the argument fits a lattice point, or not!

`point:`

- the point whose first neighbours we are looking for

`Return:`

- a vector of the first neighbours (each is given in world coordinates!)

Public Abstract Constant getLerpVertices(const point , const Toperators)

This method gets a point, and returns in a vector all of the vertices whose value influences the linear interpolation of the selected point.

`point:`

- the point whose lerp-vertices we are looking for

Public Abstract Constant getNearestPoint(const point , const Toperators)

This method can return the lattice point in whose Voronoi-region is the given point. This lattice point is the nearest neighbour point of the given point.

`point:`

- the point whose nearest neighbour we are looking for

`Toperators:`

- the operators object of type T

`Return:`

- the nearest neighbour point in the lattice in world coordinates (not in lattice locale coordinates!)

3.6. Class: IIntersectable

3.6.1. Fields

3.6.2. Properties

3.6.3. Methods

	Name and Description
Public Abstract Constant	intersect(const ray , const bounds , const T & scale , const Toperators)

Public Abstract Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector . (0D Vector)
 	
`ray:`
 	
- the intersection ray
 	
`bounds:`
 	
- the bounds between the intersection will be looked for
 	
`scale:`
 	
- scale used by ray casting algorithms
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- The collection of intersections in ascending order of dimension time.

3.7. Class: IShape

3.7.1. Fields

3.7.2. Properties

3.7.3. Methods

	Name and Description
Public Abstract Constant	fitsPoint(const pointToCheck , const Toperators)
Public Abstract Constant	intersect(const ray , const bounds , const T & scale , const Toperators)

Public Abstract Constant fitsPoint(const pointToCheck , const Toperators)

This function can decide if a point fits the Shape or not.
 	
`point:`
 	
- the point to be checked
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- true if the point fits the Shape (and false otherwise)

Public Abstract Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector . (0D Vector)
 	
`ray:`
 	
- the intersection ray
 	
`bounds:`
 	
- the bounds between the intersection will be looked for
 	
`scale:`
 	
- scale used by ray casting algorithms
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- The collection of intersections in ascending order of dimension time.

3.8. Class: Line

3.8.1. Fields

	Name and Description
Protected	dirVector
Protected	point

3.8.2. Properties

3.8.3. Methods

	Name and Description
Public Virtual Constant	fitsPoint(const pointToCheck , const Toperators)
Public Constant	getDirection ()
Public Constant	getNormal ()
Public	Line(const dirVector , const point)

Public Virtual Constant fitsPoint(const pointToCheck , const Toperators)
This function can decide if a point fits the line or not. `Return:` - true if the point fits the line (and false otherwise)

Public Constant getDirection ()
This function can return the direction vector of the line. `Return:` - the direction vector (not necessarily unit vector!)

Public Constant getNormal ()
This function can return a normal vector of the line. `Return:` - a normal vector (not necessarily unit vector!)

Public Line(const dirVector , const point)

c'tor with arguments
 	
`dirVector:`
 	
- a direction vector of the line
 	
`point:`
 	
- a point fitting the line

3.9. Class: Matrix

3.9.1. Fields

	Name and Description
Private	m
Private	n
Private	rowVects

3.9.2. Properties

3.9.3. Methods

	Name and Description
Public Virtual	Matrix ()
Public Constant	getAdj ()
Public Constant	getCol(unsigned int ind)
Public Constant	getDet ()
Public Constant	getGauss ()
Public Constant	getHeight ()
Public Constant	getInv ()
Public Constant	getMinor(unsigned int ni , unsigned int mi)
Public Constant	getRow(unsigned int ind)
Public Constant	getTransp ()
Public Constant	getWidth ()
Public Constant	isQuad ()
Public	Matrix(unsigned int n , unsigned int m)
Public	Matrix(unsigned int n , unsigned int m , T * elements)
Public	Matrix(const rows)
Public	Matrix(const other)
Public Constant	operator- ()
Public Constant	operator-(const other)
Public Constant	operator!=(const other)
Public	operator()(unsigned int ind1 , unsigned int ind2)
Public Constant	operator()(unsigned int ind1 , unsigned int ind2)
Public Constant	operator*(const other)
Public Constant	operator*(const T & scalar)
Public Constant	operator*(const vect)
Public Constant	operator*(const other)
Public	operator*=(const T & scalar)
Public Constant	operator[](unsigned int ind)
Public Constant	operator+(const other)
Public	operator+=(const other)
Public	operator=(const other)
Public	operator-=(const other)
Public Constant	operator==(const other)
Public	setCol(unsigned int ind , const newVect)
Public Virtual	setHeight(unsigned int newHeight)
Public	setRow(unsigned int ind , const newVect)
Public Virtual	setWidth(unsigned int newWidth)

3.10. Class: Plane

3.10.1. Fields

	Name and Description
Protected	normalVector
Protected	point

3.10.2. Properties

3.10.3. Methods

	Name and Description
Public Virtual Constant	fitsPoint(const pointToCheck , const Toperators)
Public Constant	getNormal ()
Public Virtual Constant	intersect(const ray , const bounds , const T & scale , const Toperators)
Public	Plane(const normalVector , const point)

Public Virtual Constant fitsPoint(const pointToCheck , const Toperators)

This function can decide if a point fits the plane or not.

`point:`

- the point to be checked

`Toperators:`

- the operators object of type T

`Return:`

- true if the point fits the plane (and false otherwise)

Public Constant getNormal ()

This function can return a normal vector of the plane.

`Return:`

- a normal vector (not necessarily unit vector!)

Public Virtual Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector . (0D Vector)

`ray:`

- the intersection ray

`bounds:`

- the bounds between the intersection will be looked for

`scale:`

- scale used by ray casting algorithms

`Toperators:`

- the operators object of type T

`Return:`

- The collection of intersections in ascending order of dimension time.

Public Plane(const normalVector , const point)

c'tor with arguments
 	
`normalVector:`
 	
- a normal vector of the plane
 	
`point:`
 	
- a point fitting the plane

3.11. Class: Point

3.11.1. Fields

	Name and Description
Protected	coords
Protected	dim

3.11.2. Properties

3.11.3. Methods

	Name and Description
Public Virtual	Point ()
Public Constant	getDim ()
Public Constant	getDistance(const other)
Public Constant	operator!=(const other)
Public	operator[](unsigned int index)
Public Constant	operator[](unsigned int index)
Public	operator=(const other)
Public Constant	operator==(const other)
Public	Point(unsigned int dim)
Public	Point(unsigned int dim , const T & fillValue)
Public	Point(unsigned int dim , const T * coord_array)
Public	Point(const other)
Public	setDim(unsigned int newDim , T data)

3.12. Class: Primitive

3.12.1. Fields

3.12.2. Properties

3.12.3. Methods

	Name and Description
Protected	Primitive ()

Protected Primitive ()
Primitive is an abstract class for geometry primitives. It shouldn't be instantied!

3.13. Class: Ray

3.13.1. Fields

3.13.2. Properties

3.13.3. Methods

	Name and Description
Public Virtual Constant	fitsPoint(const pointToCheck , const Toperators)
Public Constant	getOrigin ()
Public	Ray(const dirVector , const origin)

Public Virtual Constant fitsPoint(const pointToCheck , const Toperators)

This function can decide if a point is in the route of the ray, or not. (points in negative direction are not fitting!)

`Return:`

- true if the point fits the ray (and false otherwise)

Public Constant getOrigin ()

This function can return the origin of the ray.

`Return:`

- the origin of the ray

Public Ray(const dirVector , const origin)

c'tor with arguments

`dirVector:`

- a direction vector of the ray

`origin:`

- the origin point of the ray

3.14. Class: Surface

3.14.1. Fields

	Name and Description
Protected	material

3.14.2. Properties

3.14.3. Methods

	Name and Description
Public Constant	getMaterial ()
Public Abstract Constant	getNormal(Vector point , const T & scale , T value , const Toperators)
Public Abstract Constant	intersect(const ray , const bounds , const T & scale , const Toperators)
Public	setMaterial(const material)
Protected	Surface ()
Protected	Surface(const material)

Public Constant getMaterial ()

Getter function for the material of the surface.

`Return:`

- the material of the surface

Public Abstract Constant getNormal(Vector point , const T & scale , T value , const Toperators)

Function that can approximate a normal vector to the surface at a given point.

`point:`

- the point whose normal vector we are interested

`scale:`

- the scale parameter defines the distance from point in each dimension to build the gradient vector

`value:`

- the value at the point won't be recomputed, if it's given in this parameter

`Toperators:`

- the operators object of type T

Public Abstract Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector . (0D Vector)

`ray:`

- the intersection ray

`bounds:`

- the bounds between the intersection will be looked for

`scale:`

- scale used by ray casting algorithms

`Toperators:`

- the operators object of type T

`Return:`

- The collection of intersections in ascending order of dimension time.

Public setMaterial(const material)

Setter function for the material of the surface.

`material:`

- the new material of the surface

Protected Surface ()

default c'tor

Protected Surface(const material)

c'tor with arguments

3.15. Class: Vector

3.15.1. Fields

3.15.2. Properties

3.15.3. Methods

	Name and Description
Public Virtual	Vector ()
Public Constant	getAbs ()
Public Constant	getNormal ()
Public Constant	getUnit ()
Public Constant	multiply(const other)
Public Constant	operator- ()
Public Constant	operator-(const other)
Public Constant	operator*(const T & scalar)
Public Constant	operator*(const other)
Public	operator*=(const T & scalar)
Public Constant	operator+(const other)
Public	operator+=(const other)
Public	operator-=(const other)
Public	Vector(unsigned int dim)
Public	Vector(unsigned int dim , const T & fillValue)
Public	Vector(unsigned int dim , const T * coord_array)
Public	Vector(const other)
Public	Vector(const other)

3.16. Class: Filter

3.16.1. Fields

	Name and Description
Protected	active
Protected	filteredObject
Protected	grid

3.16.2. Properties

3.16.3. Methods

	Name and Description
Public	activate ()
Public	deactivate ()
Public	Filter ()
Public	Filter(const filteredObject , const grid)
Public Constant	getFilteredObject ()
Public Constant	getGrid ()
Public Abstract Constant	getValueAt(const point , const Toperators)
Public Constant	isActive ()
Public	setFilteredObject(const obj)
Public	setGrid(const grid)

Public activate ()
Activates this filter. (if it was already activated, the function call has no effect)

Public deactivate ()
Deactivates this filter. (if it was already deactivated, the function call has no effect)

Public Filter ()

default c'tor

Public Filter(const filteredObject , const grid)

c'tor with parameters

`filteredObject:`

- the sampleable object to be filtered

`grid:`

- the grid object that is used by this filter

Public Constant getFilteredObject ()

Getter function that returns the object to be filtered by this filter.

`Return:`

- the sampleable object

Public Constant getGrid ()

Getter function that returns the grid object that is used by this filter.

`Return:`

- the pointer of the grid object

Public Abstract Constant getValueAt(const point , const Toperators)

Function (inherited from ISampleable) that calculates the value of the sampleable object at the position specified by the argument 'point'.

`point:`

- the point where the object must calculate the return value

`Toperators:`

- the operators object of type T

`Return:`

- the value at the given point

Public Constant isActive ()

Function to query the status of the filter.

`Return:`

- true if the filter is activated (and false otherwise)

Public setFilteredObject(const obj)

Setter function for the object to be filtered.

`obj:`

- the new object to be filtered

Public setGrid(const grid)

Setter function for the grid that will be used by this filter.

`obj:`

- the pointer to the new grid object

3.17. Class: ISampleable

3.17.1. Fields

3.17.2. Properties

3.17.3. Methods

	Name and Description
Public Abstract Constant	getValueAt(const point , const Toperators)

Public Abstract Constant getValueAt(const point , const Toperators)

Function that calculates the value of the sampleable object at the position specified by the argument 'point'.
 	
`point:`
 	
- the point where the object must calculate the return value
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- the value at the given point

3.18. Class: LinearCubeFilter

3.18.1. Fields

3.18.2. Properties

3.18.3. Methods

	Name and Description
Public Virtual Constant	getValueAt(const point , const Toperators)
Public	LinearCubeFilter ()
Public	LinearCubeFilter(const filteredObject , const grid)

Public Virtual Constant getValueAt(const point , const Toperators)

Function (inherited from ISampleable) that calculates the value of the sampleable object at the position specified by the argument 'point'.
 	
`point:`
 	
- the point where the object must calculate the return value
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- the value at the given point

Public LinearCubeFilter ()
default c'tor

Public LinearCubeFilter(const filteredObject , const grid)

c'tor with parameters
 	
`filteredObject:`
 	
- the sampleable object to be filtered
 	
`grid:`
 	
- the grid object that is used by this filter

3.19. Class: LinearTetraFilter

3.19.1. Fields

3.19.2. Properties

3.19.3. Methods

	Name and Description
Public Virtual Constant	getValueAt(const point , const Toperators)
Public	LinearTetraFilter ()
Public	LinearTetraFilter(const filteredObject , const grid)

Public Virtual Constant getValueAt(const point , const Toperators)

Function (inherited from ISampleable) that calculates the value of the sampleable object at the position specified by the argument 'point'.
 	
`point:`
 	
- the point where the object must calculate the return value
 	
`Toperators:`
 	
- the operators object of type T
 	
`Return:`
 	
- the value at the given point

Public LinearTetraFilter ()
default c'tor

Public LinearTetraFilter(const filteredObject , const grid)

c'tor with parameters
 	
`filteredObject:`
 	
- the sampleable object to be filtered
 	
`grid:`
 	
- the grid object that is used by this filter

3.20. Class: MarschnerLobb

3.20.1. Fields

	Name and Description
Protected	alpha
Protected	f_m

3.20.2. Properties

3.20.3. Methods

	Name and Description
Public Constant	getAlpha ()
Public Constant	getF_m ()
Public Virtual Constant	getValueAt(const point , const Toperators)
Public	MarschnerLobb(const T & f_m , const T & alpha , const T & iso , const material)
Public	setAlpha(const T & alpha)
Public	setF_m(const T & f_m)

Public Constant getAlpha ()

This function can return the value of the alpha parameter of the Marschner-Lobb function.

`Return:`

- the value of the alpha parameter

Public Constant getF_m ()

This function can return the value of the f_m parameter of the Marschner-Lobb function.

`Return:`

- the value of the f_m parameter

Public Virtual Constant getValueAt(const point , const Toperators)

Function that calculates the value of the Marschner-Lobb function at the position specified by the argument 'point'.

`point:`

- the point where the object must calculate the return value

`Toperators:`

- the operators object of type T

`Return:`

- the value at the given point

Public MarschnerLobb(const T & f_m , const T & alpha , const T & iso , const material)

c'tor with arguments

`f\_m:`

- the F_m parameter of the Marschner-Lobb volume

`alpha:`

- the alpha parameter of the Marschner-Lobb volume

`iso:`

- the default albedo value for the volume

Public setAlpha(const T & alpha)
This function sets the alpha parameter of the Marschner-Lobb function. -: alpha parameter

Public setF_m(const T & f_m)
This function sets the f_m parameter of the Marschner-Lobb function. -: f_m parameter

3.21. Class: Volume

3.21.1. Fields

	Name and Description
Protected	iso

3.21.2. Properties

3.21.3. Methods

	Name and Description
Public Constant	getIso ()
Public Virtual Constant	getNormal(Vector point , const T & scale , T value , const Toperators)
Public Abstract Constant	getValueAt(const point , const Toperators)
Public Virtual Constant	intersect(const ray , const bounds , const T & scale , const Toperators)
Public	setIso(const T & iso)
Protected	Volume ()
Protected	Volume(const T & iso , const material)

Public Constant getIso ()
Getter function for the albedo value for the volume. `Return:` - the albedo value for the volume

Public Virtual Constant getNormal(Vector point , const T & scale , T value , const Toperators)

Function that can approximate a normal vector to the surface at a given point.

`point:`

- the point whose normal vector we are interested

`scale:`

- the scale parameter defines the distance from point in each dimension to build the gradient vector

`value:`

- the value at the point won't be recomputed, if it's given in this parameter

`Toperators:`

- the operators object of type T

Public Abstract Constant getValueAt(const point , const Toperators)

Function that calculates the value of the sampleable object at the position specified by the argument 'point'.

`point:`

- the point where the object must calculate the return value

`Toperators:`

- the operators object of type T

`Return:`

- the value at the given point

Public Virtual Constant intersect(const ray , const bounds , const T & scale , const Toperators)

Function that implements the intersection with the concrete object. If no intersection was found, it will return an empty Vector. (0D Vector)

`ray:`

- the intersection ray

`bounds:`

- the bounds between the intersection will be looked for

`scale:`

- scale used by ray casting algorithms

`Toperators:`

- the operators object of type T

`Return:`

- The collection of intersections in ascending order of dimension time.

Public setIso(const T & iso)
Setter function for the albedo value for the volume. `iso:` - the new albedo value for the volume

Protected Volume ()
default c'tor

Protected Volume(const T & iso , const material)

c'tor with argument
 	
`iso:`
 	
- the albedo value for the volume
 	
`material:`
 	
- the material of the isosurfaces of the volume (given for visualisation purposes)

3.22. Class: FloatOperators

3.22.1. Fields

	Name and Description
Private	EPS

3.22.2. Properties

3.22.3. Methods

	Name and Description
Public Virtual Constant	equals(const float & num1 , const float & num2)
Public	FloatOperators(float eps)

Public Virtual Constant equals(const float & num1 , const float & num2)

Method exams if the objects are equal or not.
 	
`num1:`
 	
- the first float number
 	
`num2:`
 	
- the second float number
 	
`Return:`
 	
- true if the two numbers are equal (and false otherwise)

Public FloatOperators(float eps)
default c'tor `eps:` - epsilon value for the equality check

3.23. Class: TypeOperators

3.23.1. Fields

3.23.2. Properties

3.23.3. Methods

	Name and Description
Public Virtual Constant	abs(const T & obj)
Public Virtual Constant	castToInt(const T & obj)
Public Virtual Constant	equals(const T & obj1 , const T & obj2)
Public Virtual Constant	greaterThan(const T & obj1 , const T & obj2)
Public Virtual Constant	greaterThanOrEquals(const T & obj1 , const T & obj2)

Public Virtual Constant abs(const T & obj)

Method returns the absolute value of the object.

`obj:`

- the object whose absolute value we are looking for

`Return:`

- the absolute value of the object

Public Virtual Constant castToInt(const T & obj)

Method casts an object of type T to an integer value.

`obj:`

- the object to be casted

`Return:`

- an integer representation of the object

Public Virtual Constant equals(const T & obj1 , const T & obj2)

Method exams if the objects are equal or not.

`obj1:`

- the first object

`obj2:`

- the second object

`Return:`

- true if the two objects are equal (and false otherwise)

Public Virtual Constant greaterThan(const T & obj1 , const T & obj2)

Method exams if the first object is greater than the second.
 	
`obj1:`
 	
- the first object
 	
`obj2:`
 	
- the second object
 	
`Return:`
 	
- true if the first object is greater than the second (and false otherwise)

Public Virtual Constant greaterThanOrEquals(const T & obj1 , const T & obj2)

Method exams if the first object is greater than the second or they are equal.
 	
`obj1:`
 	
- the first object
 	
`obj2:`
 	
- the second object
 	
`Return:`
 	
- true if the first object is greater than the second or they are equal (and false otherwise)

3.24. Class: Camera

3.24.1. Fields

	Name and Description
Protected	imgPlane
Protected	position
Protected	scale
Protected	target
Protected	u
Protected	up
Protected	v
Protected	w

3.24.2. Properties

3.24.3. Methods

	Name and Description
Public	Camera(const position , const target , const up , const imgPlane , const T & scale)
Public	capture(const modelSpace , Colour imgBuffer , unsigned short hRes , unsigned short vRes , const Toperators)
Private Constant	get3DcrossProd(const u , const v)
Public Constant	getImagePlane ()
Public Constant	getPosition ()
Public Constant	getScale ()
Public Constant	getTarget ()
Public Constant	getUpVector ()
Public	setImagePlane(const imgPlane)
Public	setPosition(const position)
Public	setScale(const T & scale)
Public	setTarget(const target)
Public	setUpVector(const up)
Private	updateCoSys ()

Public Camera(const position , const target , const up , const imgPlane , const T & scale)

c'tor with arguments

Public capture(const modelSpace , Colour imgBuffer , unsigned short hRes , unsigned short vRes , const Toperators)

Method, that creates an image of the model space from the cameras view with the given resolution.

`modelSpace:`

- the model space to be captured

`imgBuffer:`

- array where to store the image color values

`hRes:`

- the horizontal resolution of the image

`vRes:`

- the vertical resolution of the image

`Toperators:`

- the operators object of type T

`Toperators:`

- the operators object of type T

`Return:`

- the number of colour values put in the image buffer

Private Constant get3DcrossProd(const u , const v)

Cross product for 3D vectors.

`u:`

`v:`

- the two 3D vectors whose cross product we are looking for

`Return:`

- u x v

Public Constant getImagePlane ()

Getter function for the image plane object.

`Return:`

- the actual image plane object

Public Constant getPosition ()

Getter function for the position of the camera.

`Return:`

- the actual position of the camera

Public Constant getScale ()

Getter function for the scale value. This value is used by the ray casting algorithm.

`Return:`

- the actual scale value

Public Constant getTarget ()

Getter function for the target of the camera.

`Return:`

- the actual target point of the camera

Public Constant getUpVector ()

Getter function for the UP-vector of the camera.

`Return:`

- the actual UP-vector of the camera

Public setImagePlane(const imgPlane)

Setter function for the image plane object.

`imgPlane:`

- the new image plane object

Public setPosition(const position)

Setter function for the position of the camera.

`position:`

- the new position of the camera

Public setScale(const T & scale)
Setter function for the scale value. This value is used by the ray casting algorithm. `imgPlane:` - the new scale value

Public setTarget(const target)
Setter function for the target of the camera. `position:` - the new target point of the camera

Public setUpVector(const up)
Setter function for the UP-vector of the camera. `up:` - the new UP-vector of the camera

Private updateCoSys ()
This method updates the coordinate system vectors according to the other vectors. (position, target, up)

3.25. Class: Light

3.25.1. Fields

	Name and Description
Protected	intensity
Protected	position

3.25.2. Properties

3.25.3. Methods

	Name and Description
Public Constant	getIntensity ()
Public Constant	getPosition ()
Public	Light ()
Public	Light(const position , const intensity)
Public	setIntensity(const intensity)
Public	setPosition(const position)

Public Constant getIntensity ()
Get the intensity of the lightsource. `Return:` - the intensity of the lightsource (for each wavelength)

Public Constant getPosition ()
Get the position of the lightsource. `Return:` - the position of the lightsource

Public Light ()
default c'tor

Public Light(const position , const intensity)
c'tor with arguments

Public setIntensity(const intensity)
Set the intensity of the lightsource. `intensity:` - the new intensity of the lightsource

Public setPosition(const position)
Set the position of the lightsource. `position:` - the new position of the lightsource

3.26. Class: ModelSpace

3.26.1. Fields

	Name and Description
Private	background
Private	boundingVolume
Private	lights
Private	objects
Private	shader

3.26.2. Properties

3.26.3. Methods

	Name and Description
Public	addLight(const light)
Public	addObject(const object)
Public Constant	getBackground ()
Public Constant	getBoundingVolume ()
Public Constant	getColour(const ray , const T & scale , const Toperators)
Public Constant	getNearestIntersection(const ray , const T & scale , const Toperators)
Public Constant	getShader ()
Public	ModelSpace(const boundingVolume , const shader , const background)
Public	removeLight(const light)
Public	removeObject(const object)
Public	setBackground(const background)
Public	setBoundingVolume(const boundingVolume)
Public	setShader(const shader)

Public addLight(const light)
Locate a new lightsource in the model space. `light:` - the new lightsource to be located

Public addObject(const object)
Locate a new object in the model space. `object:` - the new object to be located

Public Constant getBackground ()

Getter method for the background colour.

`Return:`

- the used background colour

Public Constant getBoundingVolume ()

Getter method for the bounding volume.

`Return:`

- the used bounding volume

Public Constant getColour(const ray , const T & scale , const Toperators)

Do the ray tracing and shading for the given ray.

`ray:`

- the ray to be traced or casted

`scale:`

- the scale value used by the ray casting algorithm

`Toperators:`

- the operators object of type T

`Return:`

- the colour value in the model space for this ray

Public Constant getNearestIntersection(const ray , const T & scale , const Toperators)

This function returns the informations about the intersection with the nearest object in the ray direction in the model space.

`ray:`

- the ray to be traced or casted

`scale:`

- the scale value used by the ray casting algorithm

`Return:`

- the intersection object of the nearest intersection

Public Constant getShader ()

Getter method for the shader object.

`Return:`

- the used shader object

Public ModelSpace(const boundingVolume , const shader , const background)

c'tor with arguments

Public removeLight(const light)
Remove a lightsource from the model space. `light:` - the lightsource to be removed

Public removeObject(const object)
Remove an object from the model space. `object:` - the object to be removed

Public setBackground(const background)
Setter method for the background colour. `shader:` - the new background colour

Public setBoundingVolume(const boundingVolume)
Setter method for the bounding volume. `boundingVolume:` - the new bounding volume object

Public setShader(const shader)
Setter method for the shader object. `shader:` - the new shader object

3.27. Class: PhongShader

3.27.1. Fields

3.27.2. Properties

3.27.3. Methods

	Name and Description
Public	PhongShader ()
Public Virtual Constant	shade(const material , const light , Vector N , Vector L , Vector V , const Toperators)

Public PhongShader ()
default c'tor

Public Virtual Constant shade(const material , const light , Vector N , Vector L , Vector V , const Toperators)

Function that computes the intensity values for each colour channels according to the given parameters.
 	
`material:`
 	
- the material of the surface
 	
`light:`
 	
- the lightsource
 	
`N:`
 	
- normal of the surface
 	
`L:`
 	
- negative light direction
 	
`V:`
 	
- negative ray direction
 	
`Return:`
 	
- the intensity values for each colour channel

3.28. Class: Shader

3.28.1. Fields

3.28.2. Properties

3.28.3. Methods

	Name and Description
Public Abstract Constant	shade(const material , const light , Vector N , Vector L , Vector V , const Toperators)
Public	Shader ()

Public Abstract Constant shade(const material , const light , Vector N , Vector L , Vector V , const Toperators)

Function that computes the intensity values for each colour channels according to the given parameters.
 	
`material:`
 	
- the material of the surface
 	
`light:`
 	
- the lightsource
 	
`N:`
 	
- normal of the surface
 	
`L:`
 	
- negative light direction
 	
`V:`
 	
- negative ray direction
 	
`Return:`
 	
- the intensity values for each colour channel

Public Shader ()
default c'tor

3.29. Class: TGAImage

3.29.1. Fields

	Name and Description
Private	m_height
Private	m_pixels
Private	m_width

3.29.2. Properties

3.29.3. Methods

	Name and Description
Public Constant	getHeight ()
Public Constant	getWidth ()
Public	TGAImage ()
Public	TGAImage(const pixels , short width , short height)
Public	WriteImage(const string & filename)

4. fejezet

Documentation of the structs

4.1. Struct: Intersection

4.1.1. Fields

	Name and Description
Public	material
Public	obj
Public	position
Public	ray
Public	time
Public	value

4.1.2. Properties

4.1.3. Methods

	Name and Description
Public	Intersection ()
Public	Intersection(const obj , const position , const T & time , const ray , const T & value , const material)

Public Intersection ()
default c'tor

Public Intersection(const obj , const position , const T & time , const ray , const T & value , const material)
c'tor with arguments

4.2. Struct: Colour

4.2.1. Fields

	Name and Description
Public	a
Public	b
Public	g
Public	r

4.2.2. Properties

4.2.3. Methods

	Name and Description
Public	Colour ()
Public	Colour(unsigned char r , unsigned char g , unsigned char b , unsigned char a)

Public Colour ()
default c'tor

Public Colour(unsigned char r , unsigned char g , unsigned char b , unsigned char a)
c'tor with arguments

4.3. Struct: ImagePlane

4.3.1. Fields

	Name and Description
Public	b
Public	d
Public	l
Public	r
Public	t

4.3.2. Properties

4.3.3. Methods

	Name and Description
Public	ImagePlane ()
Public	ImagePlane(const T & d , const T & l , const T & r , const T & b , const T & t)

Public ImagePlane ()
default c'tor

Public ImagePlane(const T & d , const T & l , const T & r , const T & b , const T & t)
c'tor with arguments

4.4. Struct: Material

4.4.1. Fields

	Name and Description
Public	ka
Public	kd
Public	ks
Public	n
Public	opacity

4.4.2. Properties

4.4.3. Methods

	Name and Description
Public	Material(const ka , const kd , const ks , const float & n , const float & opacity)

Public Material(const ka , const kd , const ks , const float & n , const float & opacity)
c'tor with arguments