

# Reference Manual for MathGL3d 1.3

Jens-Peer Kuska

## Table of Contents

---

Installation of MathGL3d

Command line options

Saving Your Work

Using MathGL3d with Mathematica

MVShow3D[]

MVPutTexture[]

MVApplyTexture[]

MVGetGraphics3D[]

MVGetRasterImage[]

MVClear[]

MVClose[]

Export and Import 3D Graphics

MVLoadDXF[]

MVLoadOFF[]

MVWriteOFF[]

MVWrite3DMF[]

MVWriteDXF[]

MVWritePOVRay[]

MVWritePNG[]

MVWriteEPS[]

MVWriteVRML[]

#### Window Functions

MVReshapeWindow[]

MVShowWindow[]

MVHideWindow[]

MVIconifyWindow[]

MVSpin[]

MVStopSpin[]

#### Utilites

MVCreatePalette[]

MVRotate3D[]

MVTranslate3D[]

MVGetLightSources[]

MVGetPlotRange[]

MVGetViewPoint[]

#### Modification of Standard Functions and Packages

## Installation of MathGL3d

---

The Microsoft Windows version comes with a setup program that copies the DLLs and the executable, on Unix machines You must decompress the *Mathematica* top directory. With version 2.0 MathGL3d has a command line switch (+mvcp) to create the OpenGLViewer.m package. On Unix machines, the program must be called with the full location to the binary. If MathGL3d is called with this switch it creates the package with the `Install[LinkOpen["<mathgl-binary> -mathlink"]]` command. You must copy this package somewhere in the *Mathematica* search path for packages. Now the OpenGL viewer can be loaded with `Get["OpenGLViewer`"]`. The program comes with a OpenGLViewer.m package and you may edit the file and insert the correct location of the binary and the textures.

```
In[2]:= Get["MathGL3d`OpenGLViewer`"]

      MathGL3d installed, Version 2.0
      for Windows95 system Microsoft Windows
      © by Jens-Peer Kuska 1997,98
      with MathLink form Wolfram Research.

In[3]:= MVClose[]

Out[3]= d:/Math3.0/AddOns/ExtraPackages/MathGL3d/Binaries/Windows/
      mathview3d.exe
```

The MathGL3d package comes with files for the Online Help of *Mathematica* and you may rebuild the help index.

If you have your texture bitmaps collected in one or more directories the `$MVTexturePath` variable must be set to the list of these directories to enable the automatic search for textures with back quotes at the beginning of the file name. The best place to set the texture path is the OpenGLViewer.m package

## Command line Options

---

The OpenGL Viewer can run independent of *Mathematica*. In the stand alone mode it reads 3Scripts written by *Mathematica*, can display, animate and convert the graphics into POVRay or DXF, OFF, QuickDraw3D Metafiles, virtual reality worlds (VRML version 2.0) and PNG files. The OpenGL viewer can create smooth shaded PostScript output and is also able to read DXF files created by itself, *Mathematica* and many other programmes.

Command line switch	Default	Meaning
-mvfile <input file>	-	read 3 Script as input
-mvindx <dxfile>	-	read a DXF file as input
-mvinoff <off file>	-	read one of Geomviews OFF files
-mvpov <POVRay file>	-	save POVRay script
-mv3dm <QuickDraw3D file>	-	save QuickDraw3D meta file
-mvwrl <VRML 2.0 file>	-	save the data as VRML 2.0 world
-mvdx <DXF file>	-	save DXF file
-mvtx <png file>	-	load a PNG file as texture
-mvtxm <map type>	p	use the texture mapping of type p (lanar), c (ylindrical) s (pherical) t (orus)
±mvno	-mvno	-mvno display the data, +mvno run as filter and do not display the data from the 3 Script
±mvs	-mvs	+mvs show smooth surfaces
±mvw	-mvw	+mvw show wire frame
±mvo	-mvo	+mvo show polygon outline
±mvct	-mvct	+mvct triangulate polygons with a new center point
±mvue	+mvue	use extrusions in VRML files for lines, otherwise generate expanded polygons
-mvts <Double>	0.0	tube size
-mvta <Double>	30 °	angle to prevent line continuation
-mvps <Double>	0.0	size of spheres for point display
±mvcp	-mvcp	create the OpenGLViewer.m package and exit the program

**Notice that a command line option must be given to open the window.** When the program is called from the command line with out any option the Unix version prints out a list of possible options. The MS-Windows version opens a message window .

The command line conversion to VRML 2.0 will not create textures. The main reason is that the textures are scaled by an OpenGL function and the export is done before OpenGL

is initialized. Since OpenGL can not deal with texture dimensions  $\neq 2^n$  the PNG files given as textures must have the dimensions  $2^n \times 2^m$  where  $n$  and  $m$  are integers.

## Saving Your Work

MathGL3d supports many different output formats. It is nearly impossible to create identical output in all formats. The following table list the features supported by the exported files.

●	DXF	POVRay	Quick Draw3D	VRML 2.0	PNG Bitmaps	PostScript	PostScript Bitmap
polygons & geometry	yes	yes	yes	yes	yes	yes	
intersection of polygons	yes	yes	yes	yes	yes	no	
lines as tubes	yes	yes	yes	yes	yes	yes	
meshes	no	yes	yes	no	n/a	n/a	
surface colors	no	yes	yes	yes	yes	yes	
textures	no	partial	no	yes	yes	no	

## Using MathGL3d from Inside *Mathematica*

### ■ MVShow3D

- `MVShow3D[graphics3d, opts]` displays the `Graphics3D[ ]` object in the MathGL3d window
- `MVShow3D[surfacegraphics, opts]` displays the `SurfaceGraphics[ ]` object in the MathGL3d window
- `MVShow3D[meshgraphics3d, opts]` displays the `MeshGraphics3D[ ]` object in the MathGL3d window

The following options can be given:

MVPolygonShading	<i>symbol</i>	{MVFlat, MVSmooth, MVWireFrame}	MVSmooth	Determines how polygons are rendered As flat polygons surface or as wire
MVLineTubeSize	<i>real</i>	> 0	0.01	The thickness the lines expanded
MVTubeAngle	<i>real</i>	> 0	30.	The angle (in degree the line continuation is skipped).
MVTubeSegments	<i>integer</i>	> 0	12	The number of used to approximate tubes around
MVPointSphereSize	<i>real</i>	> 0	0.01	The radius used points as spheres
MVGrayBackground	<i>boolean</i>	{True, False}	False	Draw a light background instead of the black one.

The options of `MVApplyTexture[ ]` can also be given to `MVShow3D[ ]`, and `MVShow3D[ ]` will call `MVApplyTexture[ ]` when the `MVTexture` option is found. If a PNG file `sometex.png` is given the extension may be omitted as `"sometex`"`. When the automatic searching of the texture file in the `$MVTexturePath` variable is desired the file name can be given as `"`sometex`"`.

## ■ MVPutTexture

- `MVPutTexture[pngfile, opts]` loads a PNG bitmap into the texture memory of `MathGL3d`
- `MVPutTexture[mathGraphics, opts]` converts a *Mathematica* graphics object into a bitmap and sends it to the texture memory
- `MVPutTexture[rasterarray, opts]` send a `RasterArray[ ]` to the texture memory of `MathGL3d`

MVPutTexture[ ] returns a list of the texture identifier MVTexture2D[ *idNumber* ], the width, the height, and the type of the texture. The texture identifier can be used for texture mapping. The possible texture types returned are:

MVLuminantTexture	black white texture
MVLuminantAlphaTexture	black white texture with $\alpha$ – channel
MVRGBTexture	RGB texture
MVRGBAlphaTexture	RGB texture with $\alpha$ – channel

Possible options

are:

MVTextureSize	Automatic	The size in pixels of a <i>Mathematica</i> graphic that is used as texture . This options does not work for textures from files or RasterArrays[ ] . The default dimensions are {64, 64} .
MVDisplayGamma	1.	$\gamma$ correction for PNG files
MVClipBorderColor	True	When True the color of the border / background of a <i>Mathematica</i> graphics is removed from the texture .

The maximal possible texture size depends on the OpenGL implementation. The largest value that *must* be supported by OpenGL is 64×64, larger textures may work on certain machines.

## ■ MVApplyTexture

- MVApplyTexture[MVScene[ ] , *opts* ] maps a texture given in the options *opts* to the scene
- MVApplyTexture[MVMesh3D[ *i* ] , *opts* ] maps a texture given in the options *opts* to the mesh *i*
- MVApplyTexture[ *target* , MVTexture2D[ *k* ] , *opts* ] maps a texture MVTexture2D[ *k* ] to the scene or a mesh

If the `MVTexture` option is given `MVPutTexture[ ]` is called.

Possible options

are:

<code>MVTexture</code>	<code>None</code>	A possible texture (not already passed to <code>MathGL3d</code> a PNG – file name, a <i>Mathematica</i> graphics or a <code>RasterArray[]</code> .
<code>MVTextureMapType</code>	<code>MVPlaneMapping</code>	A possible mapping type
<code>MVTranslateTextureTarget</code>	<code>True</code>	A translation vector $\{t_x, t_y\}$ of the target to bring it into the box $\{0, 0, \_ \}, \{1, 1, \_ \}$ .
<code>MVRotateTextureTarget</code>	<code>Automatic</code>	A rotation $\{a_x, a_y, a_z\}$ , bring the target into the $\{0, 0, \_ \}, \{1, 1, \_ \}$ .
<code>MVScaleTextureTarget</code>	<code>Automatic</code>	A possible scaling $\{s_x, s_y\}$ bring the target into the $\{0, 0, \_ \}, \{1, 1, \_ \}$ .
<code>MVScaleTexture</code>	<code>Automatic</code>	A scaling $\{s_u, s_v\}$ of the texture that gives the repeated usage of the texture

Possible mapping types are:

MVPlaneMapping	The coordinates $x, y$ are used as texture coordinates.
MVCylinderMapping	The $\phi, z$ coordinates of the points in cylindrical coordinates are used
MVSphereMapping	The $\phi, \theta$ coordinates in spherical coordinates are used as texture coordinates
MVTorusMapping	The $\phi, \theta$ coordinates are used when the data points are transformed onto a torus
MVMeshUVMapping	The $u, v$ pair is used.
MVMeshVUMapping	The $v, u$ pair is used.

MVApplyTexture[ ] is used to put an additional texture onto the surface of an object inside of MathGL3d. If a texture is given for a surface it will be used to color the surface instead of the colors of the individual polygons.

### ■ MVGetGraphics3D

- MVGetGraphics3D[ ] returns the data displayed in the MathGL3d window to the kernel of Mathematica

The pure Graphics3D[ ] object as a list of points, lines, polygons is returned. The MeshGraphics3D[ ] object is not returned, instead the mesh is broken into polygons. Polygons with more than three points are broken into triangles. The MVGetGraphics3D[ ] function is useful to get the 3d data from DXF or OFF files into Mathematica.

### ■ MVGetRasterImage

- MVGetRasterImage[ ] returns the bitmap rendered by MathGL3d as a RasterArray[ ]

The function returns a RasterArray[ ] of RGBColor[ ] values. The RasterArray[ ] may be converted into an ordinary 2d graphics. If this is done the ImageSize and the AspectRatio option of the graphics object must be set to the correct values.

## ■ MVPasteGraphics

- MVPasteGraphics[ ] is an abbreviation for  
Show[Graphics[MVGetRasterImage[ ],AspectRatio->Automatic]]

## ■ MVClear

- MVClear[ ] will remove the data from the MathGL3d window and reset all values to the default ones

## ■ MVClose

- MVClose[ ] will close the MathLink connection to MathGL3d and terminate the process.

It does essentially the same as an Uninstall[ ] of the link created when calling MathGL3d in MathLink mode. The function is simply for an easy termination of MathGL3d.

## Export and Import 3D Graphics

---

### ■ MVLoadDXF

- MVLoadDXF[filename] read an AutoCAD DXF file into the MathGL3d program. To transfer the data to *Mathematica* the MVGetGraphics3D[ ] function must be used.

Only the geometric information about the points, lines and polygons is read. Layer and color information is lost.

### ■ MVLoadOFF

- MVLoadOFF[filename] read Geomviews OFF files into the MathGL3d program. To transfer the data to *Mathematica* the MVGetGraphics3D[ ] function must be used.

Only the geometry information is used, colors and normals are lost. The surface normals are typically calculated by MathGL3d itself.

## ■ MVWriteOFF

- `MVWriteOFF[filename, opts]` saves the scene as Geomview OFF file, `MeshGraphics3D[]` objects are not written.
- `MVWriteOFF[filename, graph3d, opts]` clears the MathGL3d window, shows the 3d graphics object and saves the scene as Geomview OFF file.

The colors and textures as well as the calculated normals are lost.

## ■ MVWrite3DMF

- `MVWrite3DMF[filename, opts]` saves the 3d graphics as QuickDraw3D meta file.
- `MVWrite3DMF[filename, graph3d, opts]` clears the MathGL3d window, shows the 3d graphics object and saves the 3d graphics as QuickDraw3D meta file.

Special options for the QuickDraw 3D export are:

<code>MVQuickDrawLights</code>	True	Save light positions and colors in the QuickDraw file
<code>MVQuickDrawViewPoint</code>	True	Save <i>Mathematica</i> 's view point in the file

## ■ MVWriteDXF

- `MVWriteDXF[filename, opts]` will save the 3d graphics as AutoCAD DXF file. The options are the same as for `MVShow3D[]`
- `MVWriteDXF[filename, graph3d, opts]` clears the MathGL3d window, shows the 3d graphics object and saves the 3d graphics as AutoCAD DXF file.

Notice that the information about textures, surface normals and colors are lost because the DXF format can not include this information.

## ■ MVWriteEPS

- `MVWriteEPS[filename,opt]` will save the current image in the MathGL3d window as smooth shaded encapsulated PostScript file. The only option is `EPSBitmap→True` to save the bitmap image of the MathGL3d window or the default value `EPSBitmap→False` to use the sorted polygons. The dimensions of the bitmap are the same as the window size.
- `MVWriteEPS[filename, graphics3d, opts]` clears the MathGL3d window, shows the 3d graphics object and saves the current image in the MathGL3d window as encapsulated PostScript file.

## ■ MVWritePOVRay

- `MVWritePOVRay[filename,opts]` saves the current image in the MathGL3d window as input file for the Persistence of Vision ray tracer `POVRay`.
- `MVWritePOVRay[filename, graphics3d, opts]` clears the MathGL3d window, shows the 3d graphics object and saves the current image in the MathGL3d window as script for the ray tracer.

The possible option  
is:

<code>MVTextureFilePrefix</code>	<code>mv</code>	first characters in the names of the texture bitmap files
----------------------------------	-----------------	--

## ■ MVWritePNG

- `MVWritePNG[filename,opts]` saves the current image in the MathGL3d window as portable network graphics. The dimensions of the bitmap are the same as the window size.
- `MVWritePNG[filename, graphics3d, opts]` clears the MathGL3d window, shows the 3d graphics object and saves the current image in the MathGL3d window as PNG file. The dimensions of the bitmap are the same as the window size.

Possible options  
are:

MVInterlacePNG	False	Save a interlanced PNG image
MVDisplayGamma	1.	$\gamma$ – correction for PNG files

## ■ MVWriteVRML

- `MVWriteVRML[filename, opts]` saves the 3d graphics as VRML 2.0 file.
- `MVWriteVRML[filename, graph3d opts]` clears the MathGL3d window, shows the 3d graphics object and saves the 3d graphics as VRML 2.0 file.

Special options for the VRML export  
are:

VRMLHeadLight	False	Add the head light to the scene in addition to the lights used by <i>Mathematica</i>
VRMLExtrusion	True	use the extrusion primitive for thick lines, otherwise expand the thick lines to polygons
MVTextureFilePrefix	mv	first characters in the names of the texture bitmap files

The VRML 2.0 export supports all features of MathGL3d including textures and all possible texture mappings.

## Window Functions

---

### ■ MVReshapeWindow

- `MVReshapeWindow[width, height]` sets the new width and height of the MathGL3d window. It returns the list of the old width and height.

### ■ MVShowWindow

- `MVShowWindow[ ]` reopens a hidden or iconized MathGL3d window.

## ■ MVHideWindow

- `MVHideWindow[ ]` hides the MathGL3d window.

## ■ MVIconifyWindow

- `MVIconifyWindow[ ]` iconifys the MathGL3d window.

## ■ MVSpin

- `MVSpin[angle, axis]` animates the object in the MathGL3d window. The object will rotate about the axis in steps of the angle *angle* per animation frame.

## ■ MVStopSpin

- `MVStopSpin[ ]` stops an animation started by `MVSpin[ ]` or by MathGL3d menu commands.

# Utilites

---

## ■ MVCreatePalette

- `MVCreatePalette[ ]` will create a new palette in the current notebook

The MathGL3d distribution comes with additional palettes for the export of graphics, texture options and window functions.

## ■ MVRotate3D

- `MVRotate3D[axis, angle]` will rotate the object in the MathGL3d window about the axis by the angle *angle*
- `MVRotate3D[graphics3d, axis, angle]` will rotate the *Mathematica* object about the axis by the angle *angle*

- `MVRotate3D[meshgraphics3d, axis, angle]` will rotate the `MeshGraphics3D` object about the axis by the angle *angle*

### ■ MVTranslate3D

- `MVTranslate3D[translationvector]` will translate the object in the `MathGL3d` window
- `MVTranslate3D[graphics3d, translationvector]` will translate the *Mathematica* object by the *translationvector*
- `MVTranslate3D[meshgraphics3d, translationvector]` will translate the `MeshGraphics3D` object by *translationvector*

### ■ MVGetLightSources

- `MVGetLightSources[ ]` returns the light sources used by `MathGL3d` as list  $\{\{x_p, y_p, z_p\}, \text{RGBColor}[r, g, b]\}..$ . The result can be used in the `LightSources` option of *Mathematica*

### ■ MVGetPlotRange

- `MVGetPlotRange[ ]` returns the bounding box used by `MathGL3d`

### ■ MVGetViewPoint

- `MVGetViewPoint[ ]` returns the view point used by `MathGL3d`

## Modification of Standard Functions and Packages

---

1. An additional white light source is added to the 3d lights used by *Mathematica*
2. The `Graphics`ThreeScript`` package definitions are changed to support the surface color, the `3Script` output of `SurfaceGraphics` objects is changed to keep the correct xy-range.
3. The `ParametricPlot3D[ ]` function is changed to produce `MeshGraphics3D` objects instead of `Graphics3D`.

## Copyright

---

The files in the MathGL3d distribution may be freely copied and distributed, provided that no changes whatsoever are made. All users are asked to help keep the MathView3D.w and savepng.w file consistent and "uncorrupted," identical everywhere in the world. (The CWEB system has a ``change file" facility by which users can easily make minor alterations without modifying the master source files in any way. Everybody is supposed to use change files instead of changing the files.)

The author has tried his best to produce a correct and useful program, in order to help promote computer science research and *Mathematica* but no warranty of any kind should be assumed.