

This software has been developed to help students understand and visualize geometries created by manipulating points, parametric curves and parametric surfaces.

Chapter 1

Installation

The software does not require any installation, you can simply download it and unzip in a folder of your choice. You can skip this chapter if you have no need of compiling from sources

1.1 Compiling from sources

If you have access to the sources you can compile them under Linux and OS X quite easily, since we provide with some scripts to setup everything in a mostly automated way.

Under Windows, the code has been developed and compiled under MSYS2, which is a software distro and building platform for Windows. What that means is you have a linux-like shell environment and package manager, with a mingw64-based compiler toolchain. The generated executables will run on any 64-bit Windows machine, provided you bundle a few dll files.

1.1.1 Building instructions

Both the software and its dependencies are written in C++ and use CMAKE as a build system. You can simply `git clone` all the dependencies, `cmake` install them and be good to go.

However, if you want a more guided way of building everything from scratch, you can start by cloning a repository that contains a series of scripts to help you in the process. Those will download and build all the parts with the needed `cmake` flags, using a temporary build folder so that your system root does not get polluted with any `cmake` install used in the process.

For compiling you will need the following tools:

- a recent compiler with C++14 support
- `cmake`
- the `ninja-build` build system
- `pkg-config`

You will also need some development libraries. For Debian-based linux, the list of packages is as follows:

- libglfw3-dev
 - or, as an alternative, all of the following:
libx11-dev libxrandr-dev libxinerama-dev libxcursor-dev libxi-dev
- libgl1-mesa-dev
- libgtk-3-dev

Please note that the dependency names might be slightly different depending on your distribution.

To guided compilations via provided scripts, the list of operations is as follows:

- `git clone http://github.com/francesco_cattoglio/cmake_scripts.git`
- run the `./cmake_scripts/all_git_downloads.sh` script
- `mkdir tmp_folder`
- export environment var: `export CMAKE_BUILDS_PREFIX=/full/path/to/tmp_folder`
- launch the `./cmake_scripts/build_everything.sh` script
 - this will trigger the individual build scripts for each dependency
- launch the final executable in the `./dcs/build` folder

Since the build script makes use of relative paths, please **do not change folder** before launching it.

If something goes wrong, instead of calling the `build_everything` script, you can build and install dependencies one at a time and keep the build scripts as references for the needed cmake flags.

1.1.2 Deployment

To deploy the executable, the procedure is a bit different, depending on which OS you are on. For Linux, everything is really simple since all of the libraries are compiled in as static libs.

On Windows, ...

On OS X, ...

Chapter 2

Basic Concepts

The central element of the user interface is the node graph editor. Inside it you create and link nodes of different type to assemble the scene containing all the computed objects.

There are several different kind of nodes, each with a different functionality and a different number of inputs, but only one output. This output contains some data that can be used as an input to one or more other nodes, simply by dragging a link between them using the mouse. While the same output can be used as input to multiple nodes, the opposite is an error: you cannot feed data coming from 2 outputs into one input.

One can only connect an output to an input of the same type. For example, it would not make sense to pass an **Interval** to a node that requires a **Matrix** input.

2.1 Data types

2.1.1 Geometry

The most important data type is **Geometry**. This represents a generic geometric object, it might be a point (0D), a curve (1D), a surface (2D) or a pre-fabricated mesh. Most of the time we start by creating a geometry of some sort, then manipulate it with transform functions, and in the end plug the result into a Renderer node that is the one that turns our data into something that we visualize on screen. As we will see later on, parametric transformations and sampling operations can change the geometric dimension of a Geometry object. As an example, a given curve can become a surface by applying a parametric transform to it.

2.1.2 Vector

As the name implies, the data contained in this type is just a vector with user-chosen x , y and z coordinates. There is however a very important difference between a vector and a point: a vector is **not** a Geometry and can **only** be used to build a translation Matrix, as a plane normal or for visualization purposes. This is because the vector is to be intended as a direction, not

as a generic coordinate in the 3D space. If we write a vector in homogeneous coordinates, its w component is **always zero**.

2.1.3 Matrix

This data contains a 4x4 matrix which should be interpreted as a transformation written in homogeneous coordinates. The matrix can be either parametric or a constant-valued one. See the nodes section of this chapter to know more about the different matrices that can be created.

2.1.4 Time Transform

This is a very specific 4x4 matrix with a parameter named t as in *time*. By writing time-dependent expressions, the user can attach some animation to a rendered Geometry object. This can be useful for example to show the movement of a point along a curve, or to display how a transformation progressively deforms an object into a final shape.

2.1.5 Interval and Value

Both **Interval** and **Value** data deals with parameters: the first contains the parameter and its interval boundaries, while the second contains the parameter and a specific value in its interval, to be used in a **Sample parameter** node.

2.2 Notes

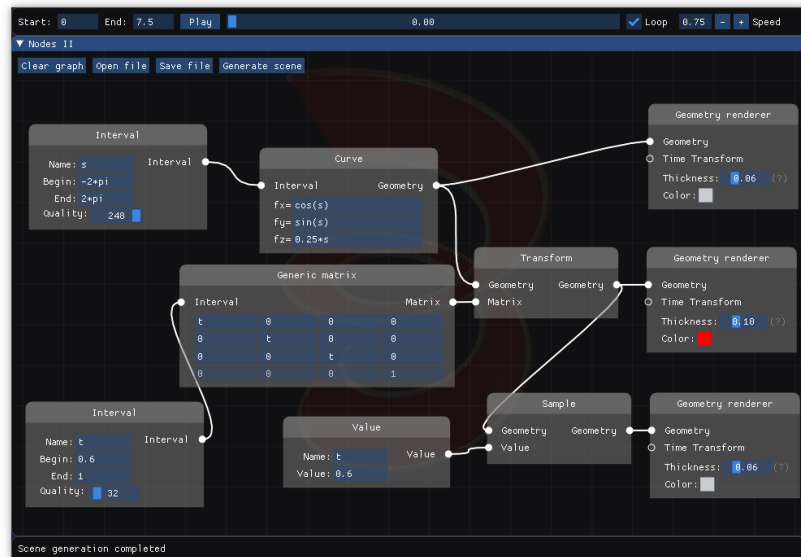
Assembling a graph does not have a 1:1 correspondency with writing procedural code (e.g.: classic C++ code). You are only describing a series of objects and a set of operations that manipulate those objects, not the order in which the computations will be executed. The order of execution will be decided by the software by looking at how nodes depend from each other. It is an error to create a loop in the graph, i.e. using the output of a transform as its own input.

Chapter 3

User Interface

The user interface consists of a top bar, a status bar in the bottom, the Scene Viewport and the Node Graph Editor with the Graph Menu Bar. When the software is loaded the Node Graph Editor is shown. On the top left of the graph editor there is an arrow that can be used to minimize it, revealing the Scene Viewport behind it. The top bar and the status bar are always visible.

3.1 Node Graph Editor



The Node Graph Editor is where most of the work is done. Each node represents an operation that creates or modifies data. Besides the inputs and the output, many nodes also have internal fields in which the user can write an expression or choose a value for a given property (e.g. a function, or $\pi/4$ for an angle).

The user can perform the following:

- To add a node, simply right-click on empty space inside the node graph editor; a popup menu will open to display all the available node types, grouped by categories. See the following section for a detailed explanation of all menu entries.
- To move a node around, left click on its header and drag it to its new position.
- You can select a group of nodes by left-clicking on empty space and dragging the mouse, or by clicking on nodes while holding the `ctrl` key, to move all of them together.
- To remove a node, right-click on its header and select “delete” from the dropdown menu.
- To connect two nodes, hover over an input or an output. When the label turns blue, click it and drag with the mouse to the label of the output or input that you want to connect to.
- To remove a connection, double-click it or hover and press the delete key
- The user can pan around the graph by right-clicking on empty space and dragging with the mouse, or by using the mouse scroll/holding the `shift` key and using the mouse scroll.
- The user can zoom in or out by holding the `ctrl` key and using the mouse scroll. clicking the mouse scroll (a.k.a. Mouse3 button) will reset the zoom level to 1.

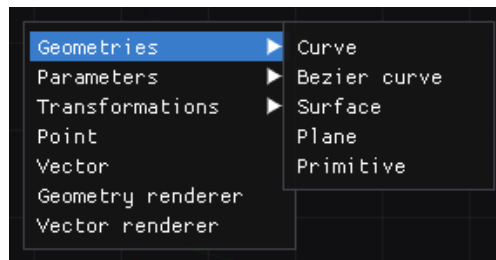


Figure 3.1: The right-click menu, with the “Geometry” submenu open

3.1.1 Geometry submenu

The geometry submenu contains the following nodes:

Curve

Inputs:	Interval
Fields:	f_x , f_y , f_z
Output:	a 1D Geometry object

This node is used to create a parametric curve using the input parameter Interval and the given expressions for fx, fy and fz. Simple to use, just make sure that the expressions are using the correct parameter name (i.e. the same name used by the input Interval)

Bezier

Inputs:	2, 3 or 4 Points (0D Geometry objects)
Fields:	Quality
Output:	a 1D Geometry object

This node uses its inputs as control points for a Bezièr curve. The degree of the curve depends on how many inputs have been connected and empty inputs are ignored. The Quality field determines the quality of the numerical approximation of the ideal Bezièr curve.

Surface

Inputs:	2 Intervals
Fields:	fx, fy, fz
Output:	a 2D Geometry object

Similarly to the curve one, this node can be used to create a parametric surface given two Intervals and the given expressions for fx, fy and fz. Both intervals are required and the user should use both of them: if one parameter is never used in the expressions, it is likely that no visual output will be produced, since a degenerated surface will be created.

Plane

Inputs:	1 point (0D Geometry), 1 Vector
Fields:	none
Output:	a Mesh Geometry object

This node takes one point and a normal vector to create a special representation of a plane. In particular, the representation consists of an evenly-spaced grid centered about the point given as input.

Please note the output is **not** a 2D Geometry, but a Mesh Geometry, this means that you will not be allowed to apply a parametric transform or a sample operation to it, only a constant-valued transform can be applied. This is because the purpose of this node is to have something useful for visualization purposes, not to have a built-in parametric plane.

Primitive

Inputs:	none
Fields:	Kind (a drop-down menu), Size
Output:	a Mesh Geometry object

This node allows the user to create a basic primitive of the given kind (Cube, Sphere, Cylinder, Cone, Pyramid or Dice). The Size fields allows adjusting the primitive dimensions. See the question mark next to the slider for a description of how this value effects the output.

Just like the Plane node, the output is a Mesh Geometry, which is useful for quick visualization or similar purposes. Parametric transforms on these objects are not allowed.

3.1.2 Parameters submenu

This submenu contains all the nodes related to parameters

Interval

Inputs:	none
Fields:	Name, Begin, End, Quality
Output:	Interval

This node defines a parameter. By assigning a name, you will then be able to use this parameter in expressions for parametric curves, surfaces and matrices. Begin and End define the closed range in which the parameter lives, while Quality allows the user to choose how many discretization points will be used when creating a curve or a surface out of this parameter.

Value

Inputs:	none
Fields:	Name, Value
Output:	Parameter Value

In a similar way to the Interval node, the Value node lets you define a parameter and assign a specific value to it. You can use it either as a “variable” to be used as input for a Matrix (e.g. define theta and then use it as a generic angle inside the matrix expressions) or as an input to the Sample Parameter node.

Sample Parameter

Inputs:	1D or 2D Geometry object, Parameter Value
Fields:	none
Output:	0D or 1D Geometry object

The parameter sampling operation allows you to “downgrade” the dimension of a parametric Geometric object, by assigning a parameter the value given as input.

In order for the operation to succeed, the Geometry must have a parameter with the same name as to the one used in the Value input (e.g. if a surface is $S(p, q)$, the input value cannot named r , only p or q will be accepted). The value must also be inside the parameter’s range. Please note that 1D Geometry created with the Bezièr curve uses a hidden parameter name, and cannot therefore be sampled.

3.1.3 Transformations submenu

This submenu contains all nodes used to define transformation matrices and to apply them to geometry objects.

Generic Matrix

Inputs:	Parameter (either Interval or Value), optional
Fields:	matrix elements
Output:	Matrix object

This node is used to define a generic Matrix, by writing one expression for each matrix element. If an Interval is given as an input, then the output matrix will be a parametric one. Note that due to the nature of the software, there is no way to define a projection matrix, since the last row of the matrix is fixed and cannot be modified.

As usual, if you provide a parameter as an input, make sure you will be actually using it inside the expressions to prevent any kind of visualization issue.

Rotation Matrix

Inputs:	none
Fields:	Axis (drop down menu), Angle
Output:	Matrix object

This node allows the user to quickly define a matrix which represents a rotation of Angle radians around the chosen Axis.

Translation Matrix

Inputs:	Vector
Fields:	none
Output:	Matrix Object

Similarly to the previous, this node allows the user to quickly define a translation matrix given the input Vector

Transform

Inputs:	Geometry object, Matrix object
Fields:	none
Output:	Geometry object

This node takes a matrix and applies it to a geometry, returning the transformed geometry. If the input matrix was a parametric matrix, then a parametric transformation will be applied. Please note that a parametric transform is not “blindly” applied to an object: if we have a parametric curve or surface that depends on the same parameter used by the matrix,

the output geometry will still be a curve or a surface, modified accordingly (e.g. applying a translation to a circle may output a spiral)

Time Transform

Inputs:	none
Fields:	matrix elements
Output:	Time Transform object

This node creates a Time Transform object, read the paragraph in the Data Types section for more informations. All the expressions for the matrix elements can contain the parameter t , i.e. “time”.

3.1.4 Point

Inputs:	none
Fields:	x, y, z
Output:	0D Geometry object

A very simple node to create a point. The implicit w coordinate is set to 1

3.1.5 Vector

Inputs:	none
Fields:	x, y, z
Output:	Vector object

A very simple node to create a Vector. As described in the data types section, a vector is to be interpreted as a “direction”, and the implicit w coordinate is set to 0

3.1.6 Geometry Renderer

Inputs:	Geometry, Time Transform
Fields:	Thickness, Color
Output:	none

This node is the one responsible for taking a Geometry object and rendering it to the screen. Every frame the Time Transform is applied to it before rendering, by evaluating the transform with a different value of t . The user can choose what color to use for any kind of Geometry, while the Thickness value only effects the display of 1D geometries (curves).

3.1.7 Vector Renderer

Inputs:	Point, Vector
Fields:	Thickness, Color
Output:	none

This node is used to display a Vector, by representing it as an arrow. Since a Vector is only a direction, the user must also provide the application point (i.e: the "tail") of the vector. Just like with the Geometry Renderer, one can choose a color and the thickness of the arrow.

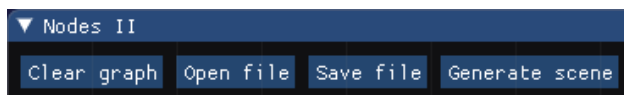
3.2 Top Bar



As it was explained before, any Geometry Renderer can have a Time Transform attached to it. This time transform contains a parameter named t that will changes in real time while the user interacts with the rendered scene, as opposed to all other parameters that are evaluated only once when the scene is generated. The Top Bar gives the user control over the value of this parameter t .

- Start: the expression for the beginning of the parameter interval. To use the constant π , write it as pi.
- End: the expression for the end of the parameter interval.
- Play/Pause: by clicking “Play” the parameter t will increment at the given speed, while by clicking “Pause” the parameter will stay frozen to the value selected by the user.
- slider: the user can click and drag the slider to manually assign a value to t
- Loop checkbox: as the name implies, if this box is ticked the animation will loop (i.e. every time t reaches the end of the interval, it will be rewinded to the start)
- Speed: the speed at which t advances when “Play” is clicked. Defaults to 0.75, meaning that 1 second of wall clock time will advance t by 0.75. The user can either write down the speed or click on the +/- signs to increase in small steps. Negative speeds are allowed!

3.3 Graph Menu Bar



The Graph Menu Bar contains 4 buttons, which function is easy to guess by their labels:

- Clear Graph: deletes all nodes and connections.

- Open File: clears the graph and loads a previously saved file.
- Save File: saves the current graph to a file.
- Generate Scene: when the user is done modifying the graph, clicking this will trigger the generation of the scene.

Clearing the graph, opening or saving a file will open a dialog asking to confirm the operation. In particular, when saving a file the extension `.toml` will be automatically added if the name does not contain it, and if any name conflict arises, then the user will be asked if the software should overwrite the file or cancel the operation.

3.4 Scene Viewport

After the scene is generated, the Graph Editor is minimized and the scene containing the generated geometries is shown. Navigation is very easy, left-click and drag to rotate the view or use the mouse scroll to zoom in and out. Currently it is not possible to pan around the scene. The lights in the scene are placed in specific points to make sure that it is easier to understand the shapes one is looking at. However, this also means that some combinations of shape and colors might produce very dark spots in some geometries.

3.5 Status Bar

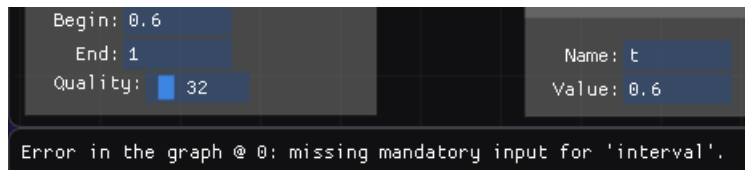


Figure 3.2: The status bar reporting an error in the user-created graph

The status bar is an often overlooked part of the User Interface, probably due to its location at the very bottom of the window. It is however extremely useful because when the software detects a mistake in the graph, it will notify the user with a (hopefully) useful message to help understand where the error is. It will also report if scene generation and file operations were successful or not.