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

1 Installation

The software does not require any installation, you can simply download it and unzip in a folder of your choice.

2 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.

2.1 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: [git cmake_scripts]. 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.

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.

2.2 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, ...

3 User Interface

3.1 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 nodes, simply by linking to them. While the same output can be used as input to multiple nodes, the opposite is an error: you cannot use data coming from 2 outputs into the same input. The data used in the graph is of one of the following types:

- `Geometry`
- `Vector`
- `Matrix`
- `Time Transform`
- `Interval`
- `Value`

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.

3.2 Data types

3.2.1 Geometry type

The most important data type is **Geometry**. This represents a generic geometric object, it might be a point, a curve, a surface 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.

3.2.2 Vector type

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**.

3.2.3 Matrix type

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.

3.2.4 Time Transform

This is a very specific 4x4 matrix with an implicit parameter named t as in *time*. By writing out 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.

3.2.5 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.

3.3 Nodes

When right-clicking on some empty space inside the node graph editor, a popup menu will open to display all the available node types, grouped by categories. Clicking on one menu item will place the newly-created node of the corresponding type.

3.3.1 Geometry > Curve

Inputs: Interval Fields: fx, fy, fz Output: a 1D Geometry object

This node is used to create a parametric curve using a parameter as defined in Interval and with the given expressions for fx, fy and fz. Simple to use, just make sure to create an expression using the correct parameter name (i.e. the same name used inside the Interval given as input)

3.3.2 Geometry > 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. Empty inputs are ignored. The Quality field determines how many points will be used to approximate the ideal Bezièr.

3.3.3 Geometry > Surface

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

Similarly to the curve node, this 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 make sure to use both of them, since if one is ignored it is likely that nothing will be visualized as a result, since that will create a degenerated surface.

3.3.4 Geometry > Plane

Inputs: one point (0D Geometry), one 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 evely-spaced grid centered about the point given as input.

Please note the output is **not** a 2D Geometry, but a Mesh Geometry, which means that the purpose of this node is to have something useful for visualization purposes, not to have a built-in parametric surface. It is not allowed to apply a parametric transform to a Mesh Geometry, only a constant-valued transform or a time-dependant transform can be applied to it.

3.3.5 Geometry > Primitive

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

This node allows the user to create a basic primitive of the given kind (Cube, Sphere, Cylinder, Cone, Pyramid or a 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. Again, only constant-valued or time-dependent transforms can be applied to this object, parametric transforms are not allowed.

3.3.6 Parameters > Interval

Inputs: None Fields: Name, Begin, End, Quality Outputs: 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.

3.3.7 Parameters > Value

Inputs: None Fields: Name, Value Outputs: 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 a theta and then use theta as the angle for a rotation matrix expressions) or as an input to the Sample Parameter node.

3.3.8 Parameters > Sample Parameter

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

The parameter sampling operation allows you to "downgrade" the dimension of a parametric Geometric object, by fixing the value of a parameter to 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 $S(p, q)$, your value cannot have the parameter name r, only p or q will be accepted). Please note that 1D Geometry created with the Bezièr curve uses a hidden parameter name, and cannot therefore be sampled.

3.3.9 Transformations > Generic Matrix

Inputs: Parameter (either Interval or Value), facultative Fields: matrix elements Outputs: 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.

3.3.10 Transformations > Rotation Matrix

Inputs: None Fields: Axis (drop down menu), Angle Outputs: Matrix object

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

3.3.11 Transformations > Translation Matrix

Inputs: Vector Fields: None Outputs: Matrix Object

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

3.3.12 Transformations > Transform

Inputs: Geometry object, Matrix object Fields: None Outputs: 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)

3.3.13 Transformations > Time Transform

Inputs: None Fields: matrix elements Outputs: Time Transform object

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

3.3.14 Point

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

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

3.3.15 Vector

Inputs: none Fields: x, y, z Outputs: 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

3.3.16 Geometry Renderer

Inputs: Geometry, Time Transform Fields: Thickness, Color Outputs: None

This node is the one responsible for taking a Geometry object and rendering it to the screen. Every frame the the Time Transform is applied to the 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).

3.3.17 Vector Renderer

Inputs: Point, Vector Fields: Thickness, Color Outputs: 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.

3.4 Notes

Assembling a graph does not have a 1:1 correspondency with writing procedural code (e.g.: classical C code). You are only describing a series of objects and a set of operations that manipulate those objects. The software will then compute the correct order in which the operations that you defined are to be executed to produce the final output.