CloudCompare
Developers course 2020

www.cloudcompare.org
Daniel Girardeau-Montaut
@CloudCompareGPL
daniel.girardeau@gmail.com

Outline

���� Architecture overview
- �� Add a simple tool to the main application

���� Plugins
- �� Create a simple plugin

���� Command line
- �� Make the plugin operation accessible via the CLI

�� Coding sprint

�� Implement a feature from the TODO list

Agenda – day 1

Day 1

10:30 – �� Architecture

overview 12:30 – Lunch

13:30 – �� Add a simple tool to the main UI

● Change the main window (Qt Designer) ●

Change the code

● Compile & run

16:00 – Coffee break

16:15 – �� Plugins (1/2)

17:30 /18:00 – End of the day

Evening: dinner?

Agenda – day 2

Day 2

- 9:00 / 9:30 – Breakfast
- 9:30 – �� Pugins (2/2)
  - Create a simple plugin
- 11:00 – Coffee break
- 11:15 – ���� Command line
  - Make the plugin operation accessible via the CLI
- 12:30 – Lunch
- 13:30 – ��Coding sprint
  - Free goal (idea from the TODO list, personal idea, etc.)
  - Alone or in group
  - Coffee break at 16:00
- 17:30 – Wrap up

Preparation

Clone the git repository: https://github.com/CloudCompare/CloudCompare.gi

t Install Qt (5.9 or higher)

Install CMake

Create the CloudCompare project with CMake, then compile and “install” it (see https://github.com/CloudCompare/CloudCompare/blob/m aster/BUILD.md)

CMake configuration

Make sure to set the CMAKE_INSTALL_PREFIX (*especially on Windows*)

You can enable at least the following plugins (*they have no dependencies*):

- GL_QEDL
- IO_QCORE
- STANDARD_QCOMPASS
- STANDARD_QHPR
- STANDARD_QM3C2
- STANDARD_QPCV
- STANDARD_QRANSAC_SD

Architecture overview

General architecture

Components

Components folder

Components license

Dependencies

Mandatory dependencies:

Qt (*CC_CORE can be compiled without if used externally*)

Optional dependencies:

CGAL: required for Delaunay 2.5D

OpenMP & TBB: for some particular parallel processes (*most of parallel processes are implemented with Qt Concurrent*)

GDAL: for raster files import/export

DXFlib: for DXF files import/export

ShapeLib: for SHP files import/export

+ all the plugins dependencies!

CC_CORE

CC_CORE

Core library, with most of the (original) algorithms

Many algorithms: octree & NN queries, distances computation, registration, segmentation, mesh & cloud sampling, statistics, etc.

Key structures: **CCVector3**, DgmOctree, Neighborhood, etc.

Basic entities: bare cloud (with SF), mesh and polyline

All used through interfaces (so as to let other tools / projects use them)

CC_CORE – cloud interfaces

- size()
- getBoundingBox()
- placeIteratorAtBeginning()
- getNextPoint()
- getPointScalarValue()
- setPointScalarValue()
- etc.

- getPoint(index)

- getPointPersistentPtr(index)

CC_CORE – other interfaces

unsigned i1, i2, i3

- size()
- getBoundingBox()
- placeIteratorAtBeginning()
- _getNextTriangle()
- etc.

- _getA(), _getB(), _getC()

- getTriangleVertIndexes()
- getTriangleVertices() •
etc.

GenericOctree, GenericProgressCallback, GenericDistribution, etc.

Simple entities

SimpleMesh + SimpleTriangle

- super minimalistic mesh entity (only 3D features) to output the Delaunay 2.5D algorithm result
  - Implements the “GenericIndexedMesh” interface

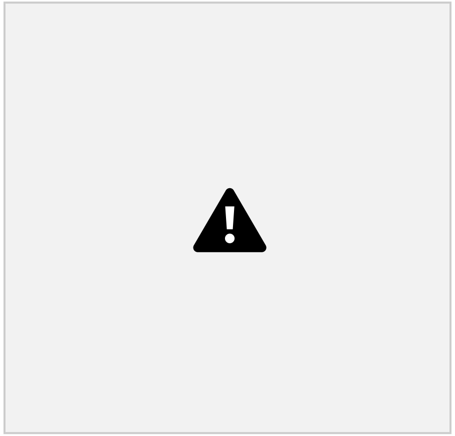
PointCloud

- simple point cloud with points and multiple scalar fields (no color or normal) used to output some algorithms results
- Implements the “GenericIndexedMesh” interface

ReferenceCloud

- “subset” cloud: list of indexes referring to a real point cloud
- **SUPER USEFUL**

CC_CORE – DgmOctree



Central octree structure

- Non-hierarchical (Morton code)
- Fast construction
- Fast NN queries
- Fixed memory cost
  - 12 bytes / per point (21 levels of subdivision)

- Key methods:
  - findNeighborsInASphereStartingFromCell (repetitive calls)
  - findNearestNeighborsStartingFromCell (repetitive calls)
  - findTheNearestNeighborStartingFromCell (repetitive calls)

    + findBestLevelFor… methods
  - getPointsInSphericalNeighbourhood (unique call)
    - +getPointsInCylindricalNeighbourhood, getPointsInBoxNeighbourhood

CC_CORE – toolboxes

- AutoSegmentationTools
- CloudSamplingTools
- DistanceComputationTools
- GeometricalAnalysisTools
- ManualSegmentationTools
- MeshSamplingTools
- PointProjectionTools
- RegistrationTools
- ScalarFieldTools
- StatisticalTestingTools

Static methods that can be applied to generic interfaces

CC_CORE – constants

PointCoordinateType

- Type of point coordinates (‘float’ by default, to minimize memory consumption)

ScalarType

- Type of scalar values (‘float’ as well)

Const values:

- PC_ONE = 1 (as *PointCoordinateType*)
- PC_NAN = Not a Number (as *PointCoordinateType)*
- NAN_VALUE = Not a Number (as *ScalarType)*

Maths:

- M_PI, M_PI_2 (=M_PI/2), SQRT_3, CC_DEG_TO_RAD, CC_RAD_TO_DEG

Quizz

What is CC_CORE library license?

How is the default “3D vector” class called?

What is the name of the “coordinates” type?

Cite 3 toolboxes of CC_CORE library

Bonus question: what is the maximum subdivision depth of the octree (*DgmOctree*)?

QCC_DB

« Q » is for … Qt

and much more

2D/3D entities

All 2D and 3D entities:

- ccPointCloud, ccMesh, ccPolyline…
- ccDrawableObject
- ccSerializableObject

- fromFile()
- toFile()
- …

Base object:
ccObject

- getClassID()
- getName()
- isEnabled()
- get/setMetaData()
- …

Base interfaces:

- isVisible()
- hasColors()
- hasNormals()
- draw()
- …

root DB object: ccHObject
(*ccHObject = groups in the DB tree*)

- getParent()
- getChildrenNumber() • getChild()
- …

+ recurvise methods

Point cloud(s)

- isShifted()
  - set/getGlobalShift()

  - visibility array
- etc.

=

- reserve()
- addPoint()

subset of points?

= out of core point cloud?

- etc.

Mesh(es)

- triangle normal, materials
- etc.

- reserve()
- addTriangle()
- etc.

- getTriGlobalIndex()
- etc.

Polyline(s)

Polylines are special clouds

• getPointPersistentPtr(index)

• getPointGlobalIndex(localIndex)

(*reference clouds* actually)

Points are implicitly connected by segments

They can be closed or not

• isClosed()
• setClosed()

They can be 2D or 3D

• isShifted()
• set/getGlobalShift()

• set2DMode()

Working with ccHObjects

In CloudCompare, all entities inserted in the database are considered as ‘ccHObject’ instances. To process an entity, it’s important to determine its type and convert it to the right type!

CC_CLASS_ENUM ccObject:: getClassID()

- Returns the class ID (= type)
- Works as a bit field (see its definition)

ccHObjectCaster

- Helper class to convert a ccHObject instance (pointer) to various types (*returns* nullptr *if it fails*)

Memory management

Memory for 3D entities features (points, colors, normals, SFs, etc.) **must be reserved beforehand**

Use the ‘reserve()’ methods (with the exact number of points) then ‘add’ the elements

```cpp
ccPointCloud* cloud = new ccPointCloud("cube");
if (!cloud->reserve(100))
{
    ccConsole::Error("Not enough memory!");
    delete cloud;
    return;
}

for (unsigned i = 0; i < 100; ++i)
{
    CCVector3 P = ...;
    cloud->addPoint(P);
}
```

Memory management (2)

- For point clouds, one has to call 'reserve' first
  - Then the other reserve methods can be called:
    - reserveTheNormsTable
    - reserveTheRGBTable
    - etc.

Almost the same thing for scalar fields:

- either create them with ccPointCloud::addScalarField (*in which case a call to ccPointCloud:: reserve or ccPointCloud::resize will change their size as well*)
- or manage the scalar field yourself and add it later

For meshes and polylines, the vertices are managed separately (**as point clouds**)

Display management

Entities can be ‘visible’ or not (will change only the visibility of the entity itself)

Entities can be ‘enabled’ or not (**will change the visibility of the entity and its children recursively**)

Entities are associated to one "display"

- ccHObject::setDisplay()

Note: in CloudCompare, the display is automatically set as the current active 3D view when calling MainWindow::addToDB() (if no display is set)

Quizz

What is the name of the helper class to convert ccHObjects to other types?

How are stored/represented the mesh and polylines vertices?

What is the name of the method to query the entity type of a ccHObject instance?

Bonus: what are the shortcuts to test whether an object is of a given ‘type’?

QCC_IO

Input/Output library

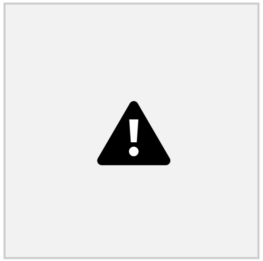
Basic mechanism to register “IO filters” to read and/or write files

- loadFile()
- saveToFile()
- **Register()**

New "IO filters" can be developed by extending the FileIOFilter interface and then "registering" them at the application start see "IO plugins"

I/O filters in the main application: BIN, ASCII, PLY, Images, DXF, SHP, rasters + QCORE_IO plugin

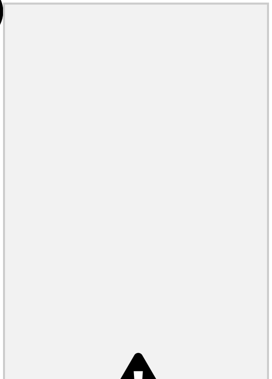
QCC_GL (& CC_FBO)



3D view (library)

ccGLWindow: OpenGL display widget

- To display a hierarchy of ccHObject* instances
  - Local database (getOwnDB())
  - Global database (getSceneDB())
- 3D background / 2D foreground
- Handles point picking (*to be*



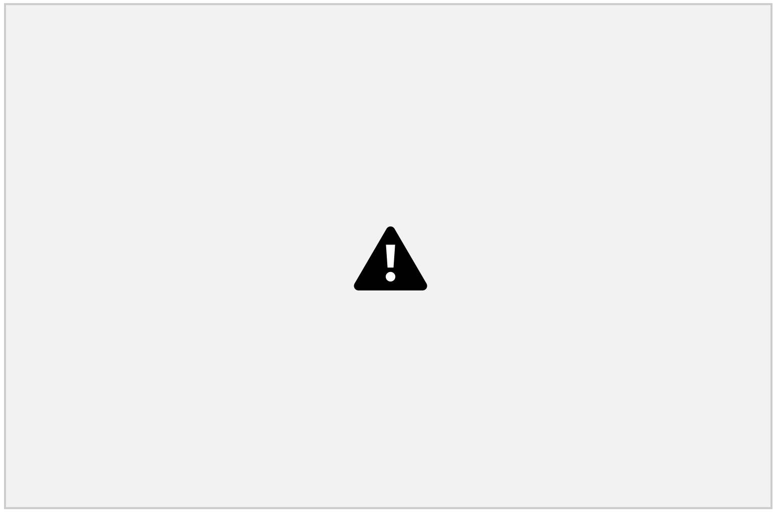
updated), stereo display, sun light + custom light, render to image/file, etc.

ccRenderingTools: SF color ramp display

ccGlFilter: shaders (see "GL" plugins such as EDL, SSAO, etc.)

CloudCompare

GUI application architecture



- addToDB()
- connectActions()
- m_selectedEntities
- m_UI
- etc.

- m_ui

- m_ui

Training

Add a new function to CloudCompare main GUI

Open Qt Designer

- Open the ‘qCC/ui_templates/mainWindow.ui’ file
- Add a new menu entry:
  - Edit > Create point
  - (rename it ‘actionCreatePoint’)
- Save the .ui file

Back to your IDE / code editor

- Declare a new slot in mainwindow.h: doActionCreatePoint()

Training

In ‘mainwindow.cpp’

- Update the MainWindow::connectActions() method
  - Add a connection between ‘m_UI->actionOpen’ and ‘MainWindow::doActionCreatePoint’

- Implement the MainWindow::doActionCreatePoint() method in the cpp file
  - Ask for the point coordinates (ccAskThreeDoubleValuesDlg)
  - Create a cloud with the corresponding point
  - Add this cloud to the main DB (addToDB())

- In the general case (*but not this time*):
  - Update MainWindow::enableUIItems as well!

Plugins

Plugins

IO plugins

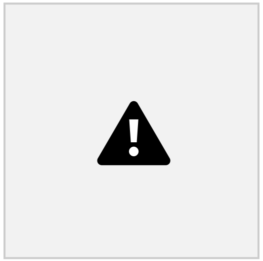
- File filters (to support new file formats)
- Simply provide an instance of “FileIOFilter”

GL plugins

- Shaders (EDL, SSAO)
- Simply provide an instance of “ccGLFilter”

Standard plugins

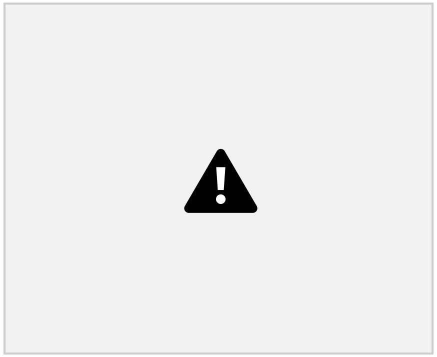
- Functions / processes
- Can have their own dialogs / 3D window / etc. • Can even define their own 3D entities (*handle with care*) • Can be accessed through the **command line interface**



Plugin files organization

\plugins

key interfaces



\plugins\3rdParty

3rd party plugins (*closed source or not, but not on the official repo.*)

\plugins\core

- \plugins\core\GL
- \plugins\core\IO
- \plugins\core\Standard

\plugins\example

Standard plugins

Standard plugins have access to:
(mostly via ccStdPluginInterface::getMainAppInterface())

- The main application data-base
  - *retrieve selected entities, retrieve any entity based on its type or unique ID, add an entity to the DB, etc.*
- The current 3D view
- The “overlay dialog” area
- The “Console” (mostly to display messages)
- etc.

Create a new plugin

Copy the “\plugins\example\ExamplePlugin” folder

- in “\plugins\3rdParty\MyPlugin”

Rename the files

- ExamplePlugin.XXX   MyPlugin.XXX

Edit “CMakeLists.txt”

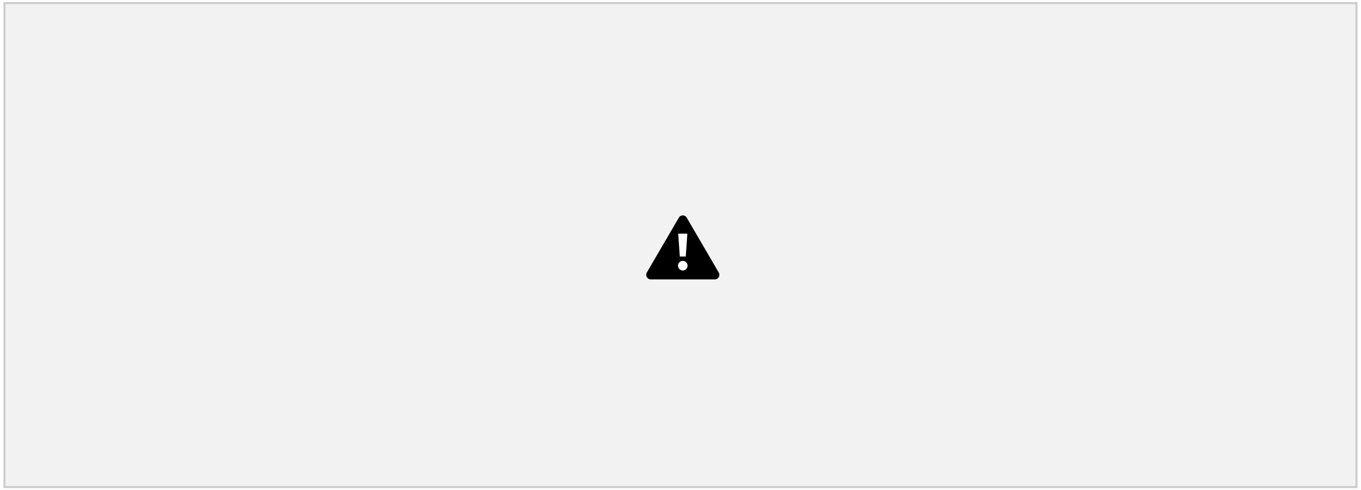
- follow the guidelines

Run CMake (on the project root) and enable your plugin (*MY_PLUGIN* or similar)

Create a new plugin (2)

Open your IDE / code editor

Follow the guidelines in the code (*renaming stuff mostly*)



We are now ready to actually implement our plugin action!

Create a new plugin (3)

Description:

- Acts on a single polyline

  onNewSelection()

- Action: smooth the selected polylines (“Chaikin” algorithm) doAction()

- Output: new polylines (with same parent as input!)
  - getMainAppInterface()::addToDB()

(the “Chaikin” algorithm code is provided ��)

Create a new plugin (4)

Adding support for the command line:

- We have to implement for our plugin “`registerCommands(ccCommandLineInterface*)`” for

- This method must return a command line interface object that will:
  - process the command line argument
  - execute the process (silently or not)
  - return the result
  - (examples: QM3C2, QPCV)

Advanced plugins walkthrough

Deep-dive in some advanced plugins to see all the possibilities:

- M3C2 or CANUPO
  - Own dialog, advanced processings, create new entities, save files, etc.
- Virtual Broom
  - Own 3D view, modifies existing entities, create new entities
- Compass
  - Register its own overlay dialogs, defines its own entities

Coding sprint

Coding sprint

Implement a feature from the TODO list … or any other suggestion!

To your votes (*we can do multiple features in parallel*)

Wrap up