Tutoriel

Version 1.00 - 26/07/2011

1 Création d'un nouveau composant (exemple : à partir du composant CGAL_Example)

Le « noyau » de Mepp ne doit en principe pas être modifié.

1.1 Votre composant doit avoir un nom en anglais (validé par Florent) et doit être rattaché à une catégorie de la liste ci-dessous :

Analysis,
Compression,
Distance,
Remeshing,
Segmentation,
Tools,
Watermarking,

ou éventuellement,

Examples.

a) Ce nom doit être également le nom du sous-dossier (contenant le code source de votre composant) au sein de cette catégorie (cf. composant CGAL_Example).

b) Le nom de la classe C++ de ce composant doit être du type **Votre_Composant**_Component (cf. composant CGAL_Example).

c) Le nom de la classe C++ du plugin (qui va de paire avec la classe ci-dessus) de votre composant doit être du type mepp_component_**Votre_Composant**_plugin (cf. composant CGAL_Example).

Important : attention à la « case sensitive » pour ces 3 points.

1.2 S'inspirer de CGAL_Example dans trunk\src\components\Examples\CGAL_Example :

Un renommage « propre » et précis tenant compte de la « case » dans les .h, .hxx et .cpp devrait vous permettre d'obtenir très facilement le squelette de votre composant.

Nous prendrons donc ci-dessous comme exemple la création du nouveau composant « Various_Tools » qui appartiendra à la catégorie « Tools ».

Pour ce faire, il faut :

a) copier le dossier « trunk\src\components\Examples\CGAL_Example » dans le dossier « trunk\src\components\Tools »,

b) renommer ce « nouveau » dossier « CGAL_Example » en « Various_Tools »,

c) supprimer récursivement tous les dossiers « .svn » du dossier « Various_Tools », (**TRES IMPORTANT**)

d) décommenter éventuellement la ligne 10 (suppression du #) du fichier « trunk\src\components\Tools\Various_Tools\cmake\use_components.txt » si vous souhaitez utiliser un autre composant au sein de votre propre composant, cf. **point 1.4** ci-dessous (ici on laisse commenté car on ne veut pas utiliser « Curvature » dans « Various_Tools »),

e) dans « trunk\src\components\Tools\Various_Tools\src », renommer tous les fichiers en changeant « CGAL_Example » par « Various_Tools » (attention à la « case sensitive »),

f) enfin, avec un éditeur de texte et une fonction « rechercher/remplacer », remplacer (en respectant la « case ») dans tous les fichiers présents (*.cpp;*.h;*.hxx) dans le dossier « trunk\src\components\Tools\Various_Tools\src » :

CGAL_Example par Various_Tools
et
CGAL_EXAMPLE par VARIOUS_TOOLS

Votre menu se déclare ensuite dans le fichier « mepp_component_Various_Tools_plugin.hxx » en le rattachant à la catégorie évoquée ci-dessus et en déclarant les actions des menus comme ci-dessous (cf. composant CGAL_Example).

```
public:
	mepp_component_CGAL_Example_plugin() : mepp_component_plugin_interface() {}
	~mepp_component_CGAL_Example_plugin()
	{
		delete actionStep_1; delete actionStep_2; delete actionStep_3;
		delete actionStep_4; delete actionStep_5; delete actionStep_6;
		delete actionStep_7; delete actionStep_8; delete actionStep_9;
	}


	void init(mainwindow* mainWindow, QList<QMdiSubWindow *> lw)
	{
		this->mw = mainWindow;
		this->lwindow = lw;
		this->mPluginName = this->metaObject()->className();

		// choice: menuTools, menuDistance_Quality_measure, menuAnalysis_Filtering,
menuSegmentation, menuRemeshing_Subdivision, menuCompression, menuWatermaking, menuExamples
		mParentMenu = mainWindow->menuExamples;

		// début --- actions ---
		actionStep_1 = new QAction(tr("Triangulate And Random Color Facets"), this);
		if (actionStep_1)
			connect(actionStep_1, SIGNAL(triggered()), this, SLOT(step1()));

		actionStep_2 = new QAction(tr("Create Center Vertex"), this);
		if (actionStep_2)
			connect(actionStep_2, SIGNAL(triggered()), this, SLOT(step2()));

		actionStep_3 = new QAction(tr("Show Black And White Facets"), this);
		if (actionStep_3)
			connect(actionStep_3, SIGNAL(triggered()), this, SLOT(step3()));

		actionStep_4 = new QAction(tr("Draw Connections"), this);
		if (actionStep_4)
			connect(actionStep_4, SIGNAL(triggered()), this, SLOT(step4()));

		actionStep_5 = new QAction(tr("Set Position And Orientation"), this);
		if (actionStep_5)
			connect(actionStep_5, SIGNAL(triggered()), this, SLOT(step5()));

		actionStep_6 = new QAction(tr("New/Add Polyhedron"), this);
```

```
		if (actionStep_6)
			connect(actionStep_6, SIGNAL(triggered()), this, SLOT(step6()));

		actionStep_7 = new QAction(tr("Load File From Component"), this);
		if (actionStep_7)
			connect(actionStep_7, SIGNAL(triggered()), this, SLOT(step7()));

		actionStep_8 = new QAction(tr("Save File From Component"), this);
		if (actionStep_8)
			connect(actionStep_8, SIGNAL(triggered()), this, SLOT(step8()));

		actionStep_9 = new QAction(tr("Sample to use Curvature component from this component"),
this);
		if (actionStep_9)
			connect(actionStep_9, SIGNAL(triggered()), this, SLOT(step9()));
		// fin --- actions ---
	}

	QList<QAction*> actions() const
	{
		return QList<QAction*>()	<< actionStep_1
						<< actionStep_2
						<< actionStep_3
						<< NULL			// menu separator
						<< actionStep_4
						<< actionStep_5
						<< NULL			// menu separator
						<< actionStep_6
						<< NULL			// menu separator
						<< actionStep_7
						<< actionStep_8
						<< NULL			// menu separator
						<< actionStep_9;
	}
```

A partir de là, le « script CMake » s'occupe de tout, il n'y a pas à toucher une seule ligne de code du noyau de Mepp (ni le « mepp_config.h.in », ni le « polyhedron_enriched_polyhedron.h », ni le « mainwindow.ui » pour le menu).
Il suffit en effet tout simplement de réinvoquer CMake (cf. readme_FR_Windows_VSxxxx.txt, readme_FR_Linux.txt ou readme_FR_Mac_OS_X.txt).

1.3 Commenter votre nouveau composant avec Doxygen

Cf. mini-tutoriel pour Doxygen : Franck Hecht - http://franckh.developpez.com/tutoriels/outils/doxygen/
(pas exhaustif mais dans l'ensemble suffisant).

1.4 Utiliser le code d'un composant X au sein de votre composant (sans la gestion des boîtes de dialogue du composant X pour le moment) :

Note : dans cet exemple, on utilise le composant Curvature dans le composant CGAL_Example.

a) Dans « trunk\src\components\Examples\CGAL_Example\cmake\use_components.txt », renseigner les fichiers *.c/*.cpp dont vous avez besoin avec la fonction **set** de CMake (cf. dernière ligne) :

```
set( use_components ../../Analysis/Curvature/src/Curvature_Component.cpp ../../Analysis/Curvature/src/extract_Vpropres.cpp )
```

b) Dans « trunk\src\components\Examples\CGAL_Example\mepp_component_CGAL_Example_plugin.cpp », renseigner en haut du fichier les lignes suivantes :

```
// we want to use Curvature component
#include "../../../Analysis/Curvature/src/Curvature_Component.h"
typedef boost::shared_ptr<Curvature_Component> Curvature_ComponentPtr;
// we want to use Curvature component
```

c) Ensuite, l'appel et l'utilisation de Curvature dans « trunk\src\components\Examples\CGAL_Example\mepp_component_CGAL_Example_plugin.cpp », se fait comme ceci, cf. fonction step9() :

```
void mepp_component_CGAL_Example_plugin::step9()
{
	// active viewer
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

		QApplication::setOverrideCursor(Qt::WaitCursor);

		// we use CGAL_Example component here (as usual)
		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);
		component_ptr->TriangulateAndRandomColorFacets(polyhedron_ptr);

		component_ptr->set_init(2);
		// we use CGAL_Example component here (as usual)

		// we use Curvature component here
		bool IsGeo;
		double radius;
		Curvature_ComponentPtr component_ptr_curvature =
findOrCreateComponentForViewer<Curvature_ComponentPtr, Curvature_Component>(viewer, polyhedron_ptr);

		// params
		radius = 0.001;
		IsGeo=true;

		mw->statusBar()->showMessage(tr("Curvature..."));
		component_ptr_curvature->principal_curvature(polyhedron_ptr,IsGeo,radius);
		mw->statusBar()->showMessage(tr("Curvature is done"));

		component_ptr_curvature->set_init(2);

		component_ptr_curvature->ConstructColorMap(polyhedron_ptr,1);
		viewer->recreateListsAndUpdateGL();
		// we use Curvature component here
	}

	QApplication::restoreOverrideCursor();
}
```

2 Descriptions des fichiers d'un composant (exemple : CGAL_Example)

CGAL_Example_Polyhedron.h

Regroupe toutes les déclarations et tous les typedefs concernant CGAL et le polyèdre. Comme il inclut le polyèdre, il suffit d'inclure ce fichier n'importe où dans le composant pour avoir accès au polyèdre et aux typedefs déclarés. Il est donc conseillé d'inclure ce fichier et pas directement le fichier « polyhedron.h » de Mepp.

CGAL_Example_Items.h

Définis les items propres au composant. Permet de rajouter des variables et des méthodes aux sommets, arêtes et facettes du polyèdre, et au polyèdre lui-même.

CGAL_Example_Component.h/.cpp

Définis l'API (interface de programmation) du composant. C'est l'ensemble des fonctions présentées aux développeurs de la plateforme. Le composant est utilisé exclusivement au travers de cette API. Aucun élément d'interface graphique ne doit être présent à l'intérieur de ces fichiers, ni de leurs fichiers dépendants.

mepp_component_CGAL_Example_plugin.h/.cpp

Fait le lien entre l'interface graphique et le composant. Tous les éléments d'interface graphique doivent se trouver à l'intérieur de ces fichiers (et de leurs fichiers dépendants si besoin). Communique avec le composant à travers l'API définie dans « CGAL_Example_Component.h ».

3 Explication d'un composant en détail (exemple : CGAL_Example)

Fichiers du composant (plugin, ie. librairie dynamique) : mepp_component_CGAL_Example_plugin.h/.cpp
Fichiers du composant (partie CGAL pure) : CGAL_Example_Component.h/.cpp

3.1 Méthodes communes à tous les composants (cf. mepp_component_CGAL_Example_plugin.cpp) :

`void mepp_component_CGAL_Example_plugin::pre_draw()`, méthode appelée automatiquement avant le rendu du polyèdre. Il est possible de rajouter le rendu d'éléments spécifiques au composant. Toutes les méthodes pre_draw() sont appelées avant le rendu. Il est conseillé de permettre d'activer/désactiver le rendu pre_draw (quand celui-ci n'est pas vide) à travers une ou plusieurs variables liées à l'interface graphique (appui sur un bouton, clic sur un élément du menu).

`void mepp_component_CGAL_Example_plugin::post_draw()`, méthode appelée automatiquement après le rendu du polyèdre. Il est possible de rajouter le rendu d'éléments spécifiques au composant. Toutes les méthodes post_draw() sont appelées après le rendu. Il est conseillé de permettre d'activer/désactiver le rendu post_draw (quand celui-ci n'est pas vide) à travers une ou plusieurs variables liées à l'interface graphique (appui sur un bouton, clic sur un élément du menu).

`void mepp_component_CGAL_Example_plugin::pre_draw_all_scene()`, idem que pre_draw(), mais appelée automatiquement avant le rendu de TOUS les polyèdres (uniquement avec le mode « Space » de Mepp).

`void mepp_component_CGAL_Example_plugin::post_draw_all_scene()`, idem que post_draw(), mais appelée automatiquement après le rendu de TOUS les polyèdres (uniquement avec le mode « Space » de Mepp).

`void mepp_component_CGAL_Example_plugin::OnMouseLeftDown(QMouseEvent *event)`, méthode appelée suite à un clic gauche de la souris lorsque le bouton est enfoncé et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnMouseLeftUp(QMouseEvent *event)`, méthode appelée suite à un clic gauche de la souris lorsque le bouton est relâché et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnMouseRightDown(QMouseEvent *event)`, méthode appelée suite à un clic droit de la souris lorsque le bouton est enfoncé et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnMouseRightUp(QMouseEvent *event)`, méthode appelée suite à un clic droit de la souris lorsque le bouton est relâché et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnMouseMotion(QMouseEvent *event)`, méthode appelée suite à un mouvement de la souris et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnMouseWheel(QWheelEvent *event)`, méthode appelée suite à un mouvement de la molette de la souris et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnKeyPress(QKeyEvent *event)`, méthode appelée suite à un appui sur une touche du clavier lorsque celle-ci est enfoncée et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

`void mepp_component_CGAL_Example_plugin::OnKeyRelease(QKeyEvent *event)`, méthode appelée suite à un appui sur une touche du clavier lorsque celle-ci est relâchée et que la touche 'Alt' (sous Windows et Mac OS X) ou 'Drapeau Windows' (sous Linux) est maintenue.

3.2 Méthodes communes à tous les composants (cf. CGAL_Example_Component.cpp) :

`CGAL_Example_Component::CGAL_Example_Component(Viewer* v, PolyhedronPtr p)`, constructeur du composant. Todo : componentName et init.

3.3 Méthodes spécifiques au composant CGAL_Example :

3.3.1 Step1 : triangulation du maillage et « tirage » au sort de la couleur des facettes

Dans CGAL_Example_Component.h :

Nous allons tout d'abord déclarer la fonction relative à notre première étape. Cette fonction sera « publique ».

```
void TriangulateAndRandomColorFacets(PolyhedronPtr pMesh);
```

Le corps de cette fonction sera quant à lui présent dans CGAL_Example_Component.cpp :

```
void CGAL_Example_Component::TriangulateAndRandomColorFacets(PolyhedronPtr pMesh)
{
	pMesh->triangulate(); (1)

	srand((unsigned)time(NULL));

	Facet_iterator pFacet = NULL; (2)
```

```
	for (pFacet = pMesh->facets_begin(); pFacet != pMesh->facets_end(); pFacet++)
	{
		float rand255r = (float) rand() / (float) RAND_MAX;
		float rand255g = (float) rand() / (float) RAND_MAX;
		float rand255b = (float) rand() / (float) RAND_MAX;

		pFacet->color(rand255r, rand255g, rand255b);
	}
}
```

Explications :

(1) Le polyèdre est tout d'abord « triangulé » dans tous les cas. Cette opération a un effet seulement si le polyèdre n'est pas déjà triangulé.
(2) On parcourt toutes les facettes grâce à un itérateur de facettes, et pour chaque facette on attribue des composantes R, V, B aléatoires.

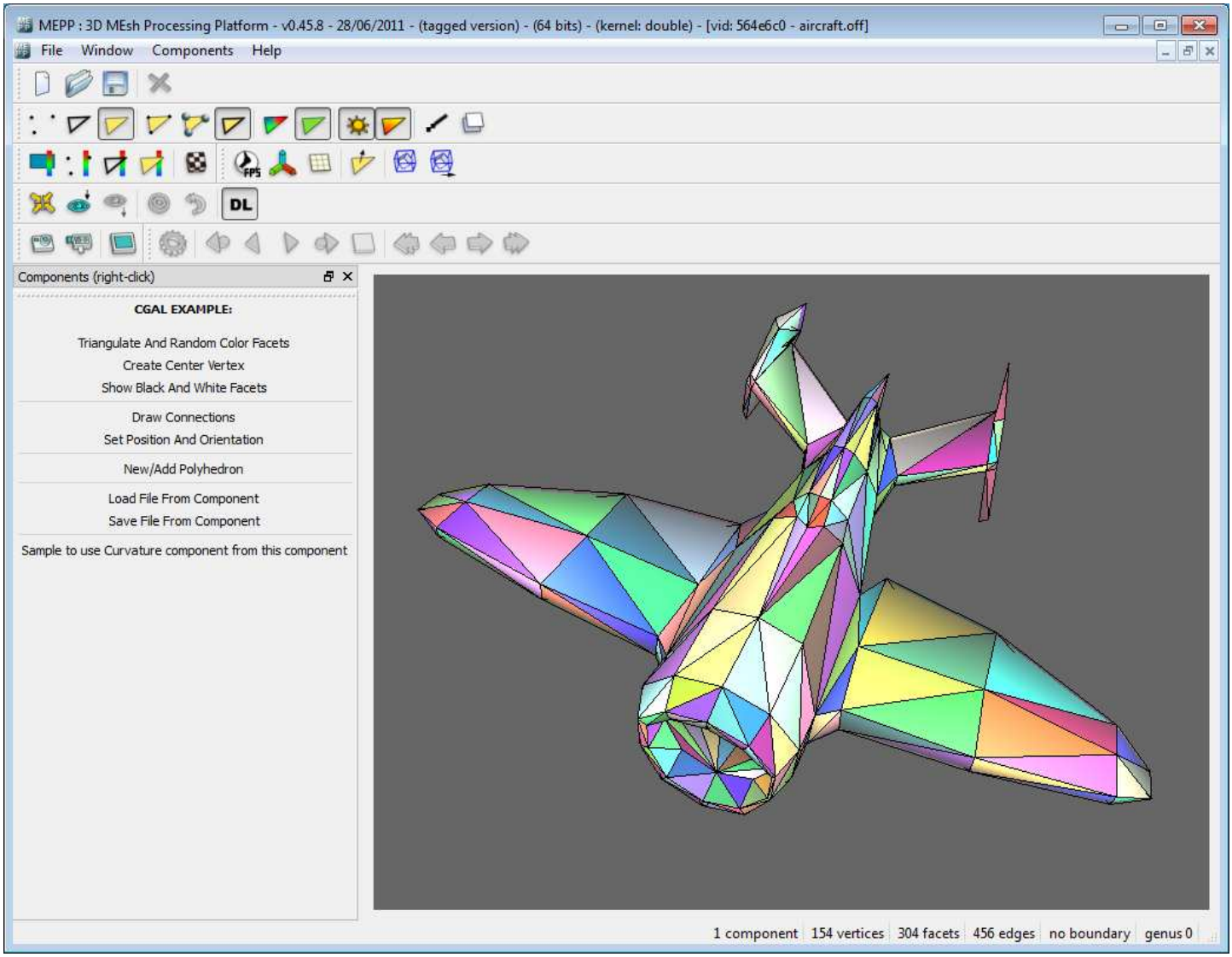
Dans mepp_component_CGAL_Example_plugin.cpp :

On appelle notre première fonction ci-dessus par le biais de la fonction « step1() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step1()
{
	QApplication::setOverrideCursor(Qt::WaitCursor); (3)

	// active viewer
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron(); (4)

		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr); (5)
		component_ptr->TriangulateAndRandomColorFacets(polyhedron_ptr); (6)

		component_ptr->set_init(2); (7)
		viewer->recreateListsAndUpdateGL();
	}

	QApplication::restoreOverrideCursor(); (8)
}
```

Explications :

(3) On signale à l'utilisateur que Mepp est actuellement occupé par le biais d'un curseur de souris « sablier ».
(4) On récupère un pointeur sur le polyèdre.
(5) On récupère un pointeur sur le composant.
(6) On appelle notre fonction en lui passant notre polyèdre en pointeur.
(7) On passe le composant à l'état 2 (il était à l'état 1 suite à sa création, cf. point **3.2** ci-dessus) et on rafraichit l'affichage.
(8) On signale à l'utilisateur que le traitement est désormais terminé en repositionnant le curseur « classique » de la souris.



3.3.2 Step2 : création d'un point au centre des facettes (et donc de 3 sous-facettes) dont la couleur est proche d'une couleur choisi par l'utilisateur

Dans CGAL_Example_Component.h :

Nous allons tout d'abord déclarer la fonction relative à notre deuxième étape. Cette fonction sera « publique ».

```
void CreateCenterVertex(PolyhedronPtr pMesh, bool save);
```

Le corps de cette fonction sera quant à lui présent dans CGAL_Example_Component.cpp :

```
void CGAL_Example_Component::CreateCenterVertex(PolyhedronPtr pMesh)
{
	CSubdivider_sqrt3<Polyhedron,Enriched_kernel> subdivider;

	long dist, maxdist=0, mindist=ColourDistance(0, 0, 0, 255, 255, 255), mindist_pct;

	Facet_iterator pFacet = NULL; (1)
	for (pFacet = pMesh->facets_begin(); pFacet != pMesh->facets_end(); pFacet++)
	{
		pFacet->tag(0);

		dist = ColourDistance((unsigned char)round(color(0)*255.), (unsigned
char)round(color(1)*255.), (unsigned char)round(color(2)*255.), (unsigned char)round(pFacet-
>color(0)*255.), (unsigned char)round(pFacet->color(1)*255.), (unsigned char)round(pFacet-
>color(2)*255.));

		if (dist < mindist)
			mindist = dist;
```

```
		if (dist > maxdist)
			maxdist = dist;
	}

	int pct=3;
	mindist_pct=((maxdist-mindist)*pct/100)+mindist; (2)

	int cpt = 1;
	for (pFacet = pMesh->facets_begin(); pFacet != pMesh->facets_end(); pFacet++)
	{
		if (pFacet->tag()==0)
		{
			dist = ColourDistance((unsigned char)round(color(0)*255.), (unsigned
char)round(color(1)*255.), (unsigned char)round(color(2)*255.), (unsigned char)round(pFacet-
>color(0)*255.), (unsigned char)round(pFacet->color(1)*255.), (unsigned char)round(pFacet-
>color(2)*255.));

			if ((dist >= mindist) && (dist <= mindist_pct))
			{
				pFacet->tag(1);
				Vertex_handle hVertex = subdivider.create_center_vertex(*pMesh, pFacet)-
>vertex(); (3)

				float rand255r = (float) rand() / (float) RAND_MAX;
				float rand255g = (float) rand() / (float) RAND_MAX;
				float rand255b = (float) rand() / (float) RAND_MAX;
				hVertex->color(rand255r, rand255g, rand255b);
			}
		}
	}
}
```

Explications :

(1) On parcourt toutes les facettes grâce à un itérateur de facettes afin de calculer les distances de couleur « min » et « max » par rapport à la couleur choisi par l'utilisateur.
(2) On parcourt toutes les facettes grâce à un itérateur de facettes. Pour toutes les facettes non préalablement découpées en 3 sous-facettes (donc non taguées) proches en terme de distance de couleur de 3% de la couleur choisi par l'utilisateur, on crée le point central et les 3 sous-facettes.
(3) Le point central ainsi que les 3 sous-facettes sont créés grâce à un objet de la classe CSubdivider_sqrt3.

Dans mepp_component_CGAL_Example_plugin.cpp :

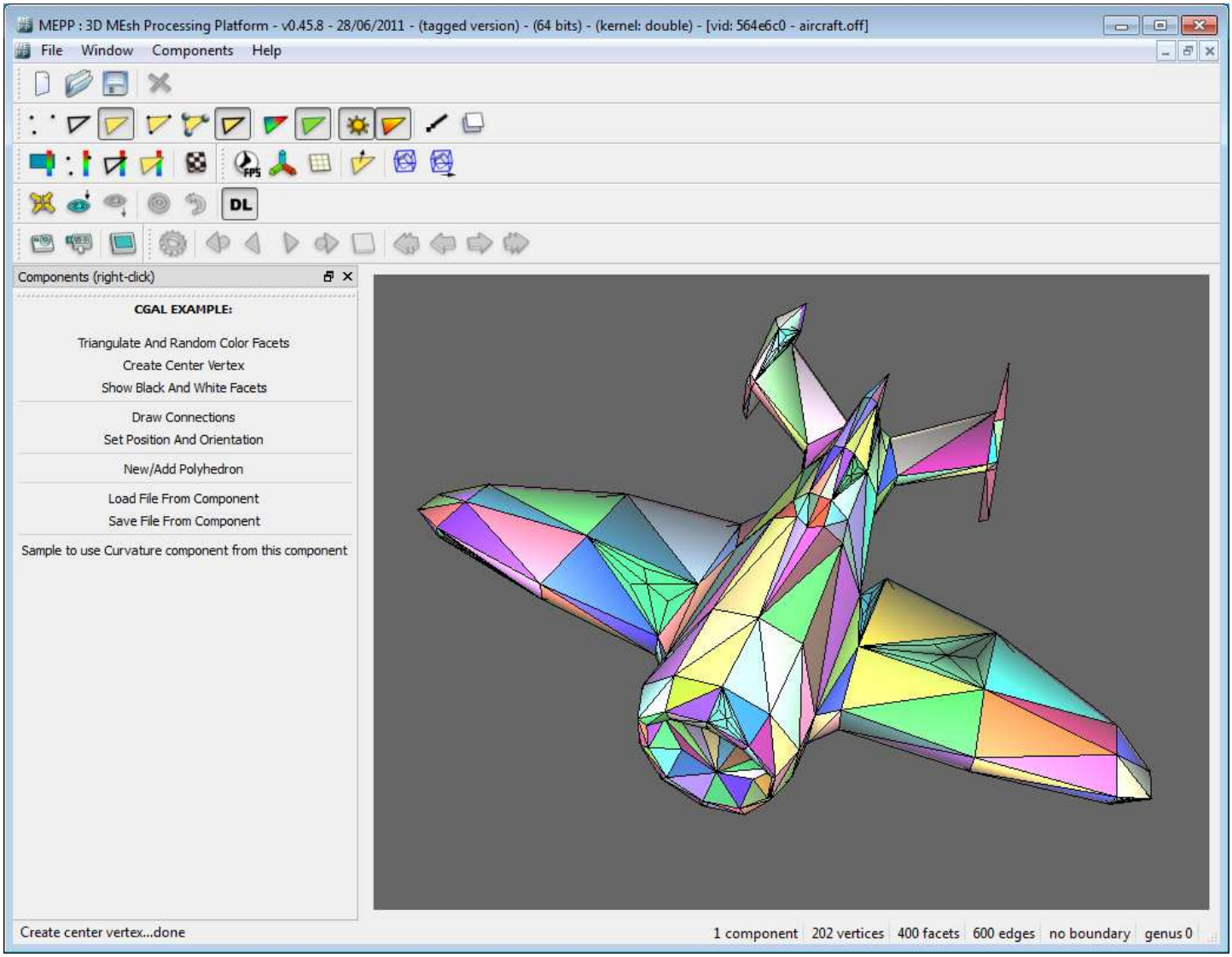
On appelle notre deuxième fonction ci-dessus par le biais de la fonction « step2() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step2()
{
	// active viewer
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron(); (4)

		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr); (5)

		QColor current_color(int(component_ptr->color(0)*255.), int(component_ptr-
>color(1)*255.), int(component_ptr->color(2)*255.));
		QColor new_color = QColorDialog::getColor(current_color, viewer); (6)

		if (new_color.isValid())
		{
			component_ptr->color(float(new_color.red())/255., float(new_color.green())/255.,
float(new_color.blue())/255.);
```

```
                    SettingsDialog_CGAL_Example dial;
                    if (dial.exec() == QDialog::Accepted) (7)
                    {
                           QApplication::setOverrideCursor(Qt::WaitCursor);

                           int iteration = dial.Iteration->value();

                           mw->statusBar()->showMessage(tr("Create center vertex...")); (8)

                           for (int p=0; p<viewer->getScenePtr()->get_nb_polyhedrons(); p++) (9)
                                  for (int i=0; i<iteration; i++)
                                         component_ptr->CreateCenterVertex(viewer->getScenePtr()-
>get_polyhedron(p), false);

                           mw->statusBar()->showMessage(tr("Create center vertex...done"));

                           viewer->recreateListsAndUpdateGL();

                           QApplication::restoreOverrideCursor();
                           return;
                    }
             }

             mw->statusBar()->showMessage(tr("Create center vertex...canceled"));
      }

      QApplication::restoreOverrideCursor();
}
```

Explications :

(4) On récupère un pointeur sur le polyèdre.
(5) On récupère un pointeur sur le composant.
(6) On appelle la boîte de dialogue de choix de couleur de Qt et on récupère la couleur choisi.
(7) On appelle notre boîte de dialogue créée à l'aide Qt Designer (cf. point **4** ci-dessous) afin de connaitre le nombre de fois que l'utilisateur souhaite créer des points centraux (et donc des sous-facettes) : iteration.
(8) On renseigne la barre de statut de Mepp afin de signifier qu'un traitement est en cours.
(9) Pour tous les polyèdres, on appelle notre fonction (iteration fois) en lui passant notre polyèdre en pointeur.



3.3.3 Step3 : pour toutes les facettes découpées à l'étape 2, parcours de toutes les facettes « voisines » présentant une demi-arrête commune et attribution d'une couleur blanche ou noire à ces dernières en fonction d'un critère donné…

Dans CGAL_Example_Component.h :

Nous allons tout d'abord déclarer la fonction relative à notre troisième étape. Cette fonction sera « publique ».

```
void ShowBlackAndWhiteFacets(PolyhedronPtr pMesh);
```

Le corps de cette fonction sera quant à lui présent dans CGAL_Example_Component.cpp :

```
void CGAL_Example_Component::ShowBlackAndWhiteFacets(PolyhedronPtr pMesh)
{
	Facet_iterator pFacet = NULL; (1)
	for (pFacet = pMesh->facets_begin(); pFacet != pMesh->facets_end(); pFacet++)
	{
		int all_tags_are_one;

		if (pFacet->tag()==1)
		{
			// circulate around facet
			Halfedge_around_facet_circulator he, end;
			he = end = pFacet->facet_begin();
			all_tags_are_one=1;

			CGAL_For_all(he, end) (2)
			{
```

```
                        if (he->opposite()->face() != NULL)
                        {
                                if (he->opposite()->face()->tag()==0) (3)
                                {
                                        he->opposite()->face()->color(1., 1., 1.);
                                        all_tags_are_one=0;
                                }
                        }
                }
                if (all_tags_are_one) (4)
                        pFacet->color(0., 0., 0.);
        }
    }
}
```

Explications :

(1) On parcourt toutes les facettes grâce à un itérateur de facettes.
(2) Pour toutes les facettes découpées à l'étape 2 (et donc taguées), on utilise un circulateur de demi-arrêtes et pour toutes ces demi-arrêtes,
(3) on colorie en blanc la facette incidente (si elle existe !) à la demi-arrête opposée s'il ne s'agit pas d'une facette taguée, sinon,
(4) on colorie en noir la facette incidente (si elle existe !) à la demi-arrête opposée si toutes les facettes « voisines » sont taguées.

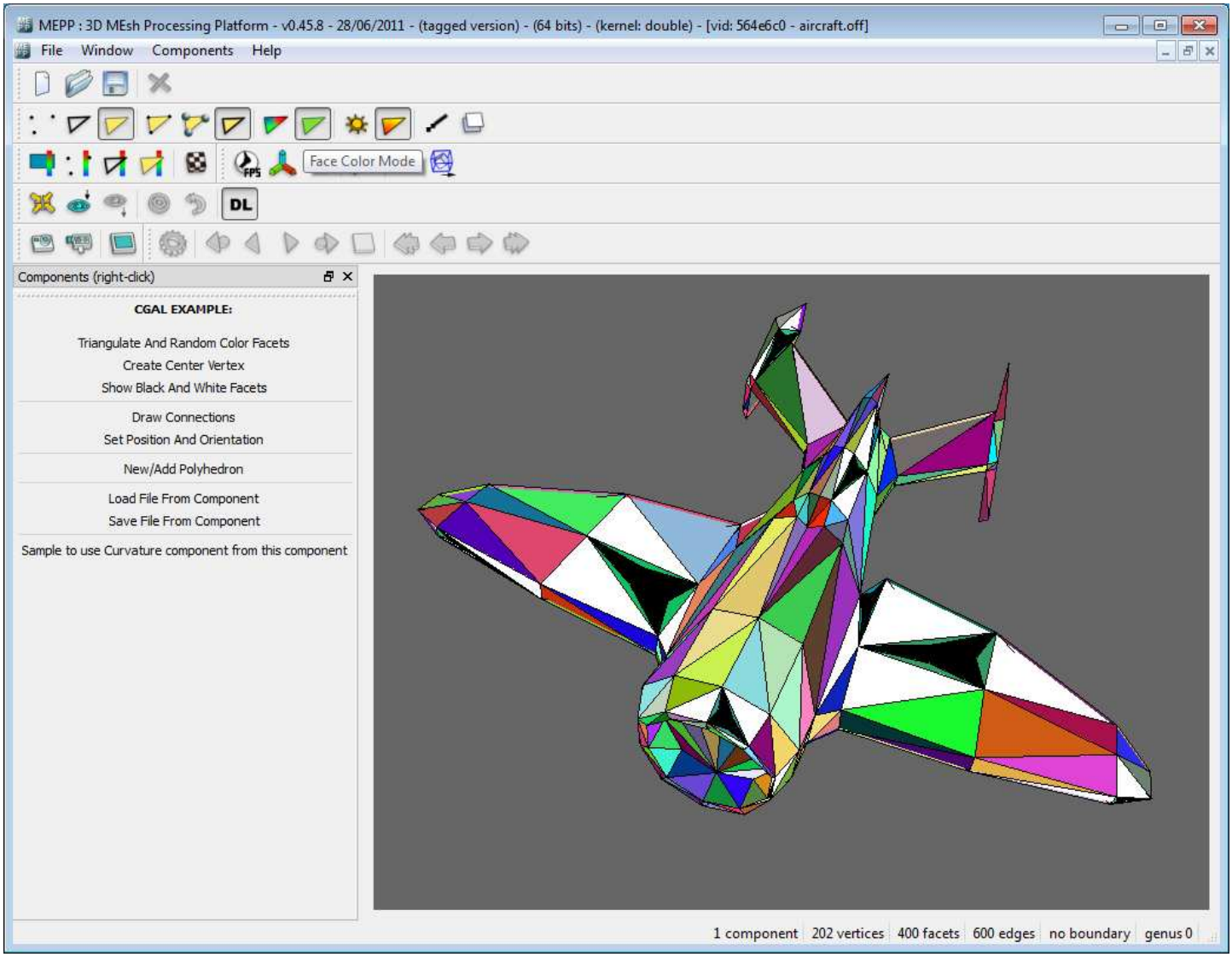
Dans mepp_component_CGAL_Example_plugin.cpp :

On appelle notre troisième fonction ci-dessus par le biais de la fonction « step3() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step3()
{
    QApplication::setOverrideCursor(Qt::WaitCursor);

    // active viewer
    if (mw->activeMdiChild() != 0)
    {
        Viewer* viewer = (Viewer *)mw->activeMdiChild();
        PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

        CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);
        component_ptr->ShowBlackAndWhiteFacets(polyhedron_ptr); (5)

        viewer->recreateListsAndUpdateGL();
    }

    QApplication::restoreOverrideCursor();
}
```

Explications :

(5) On appelle notre fonction en lui passant notre polyèdre en pointeur.



3.3.4 Step4 : afficher des « connexions » entre plusieurs polyèdres (uniquement avec le mode « Space » de Mepp)

Dans mepp_component_CGAL_Example_plugin.cpp :

On appelle notre quatrième fonction ci-dessus par le biais de la fonction « step4() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step4()
{
	QApplication::setOverrideCursor(Qt::WaitCursor);

	// active viewer
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);

		component_ptr->set_init(3); (1)
		viewer->updateGL();
	}

	QApplication::restoreOverrideCursor();
```

```
}
```

Explications :

(1) On positionne le flag « init » à 3 (à la création du composant le flag « init » est initialisé à 1).

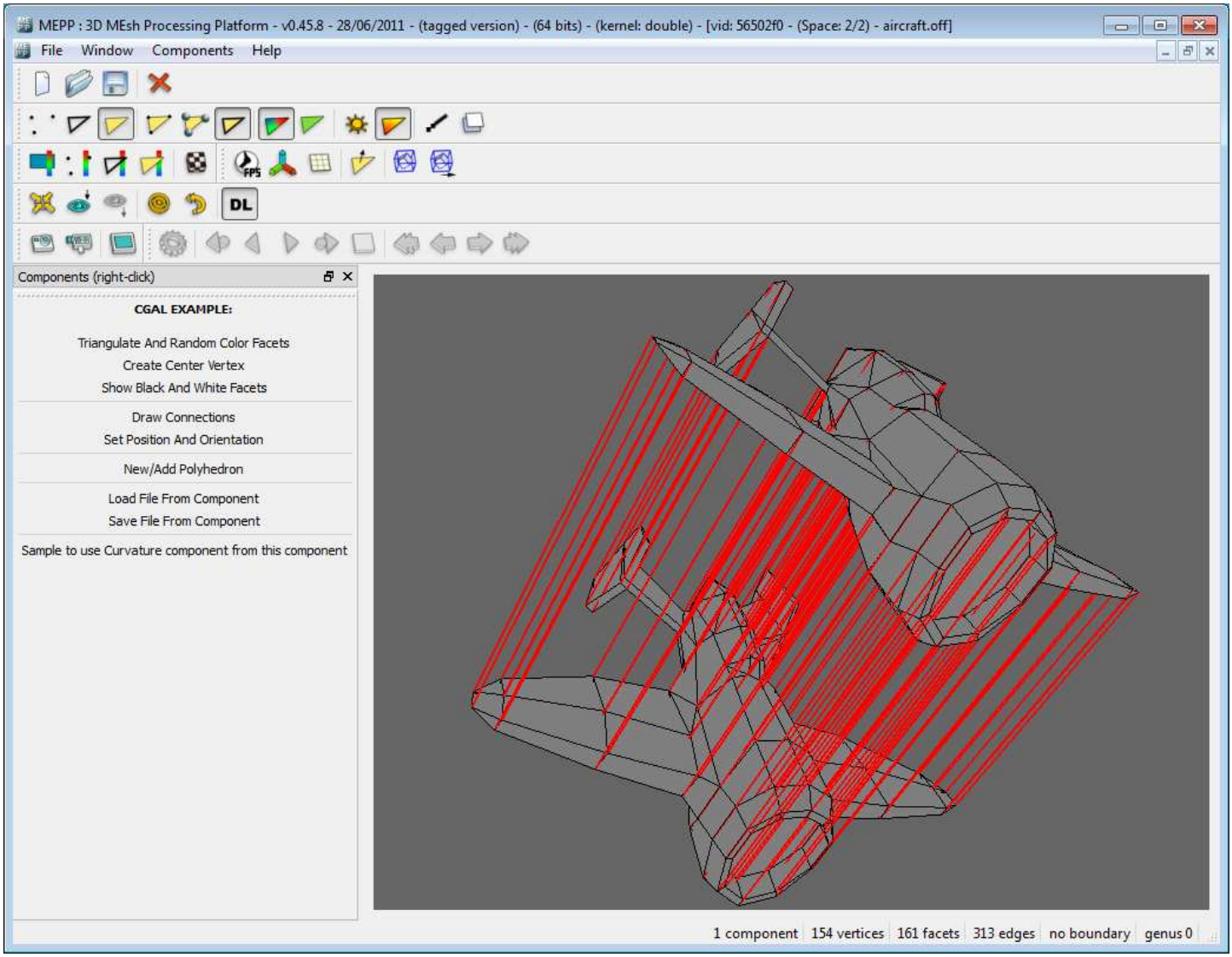
Note : le flag « positionné » à 2 sert pour une autre fonction.

Ensuite, la fonction `void mepp_component_CGAL_Example_plugin::post_draw_all_scene()` (cf. **point 3.1**) appelée automatiquement après le rendu de TOUS les polyèdres (uniquement avec le mode « Space » de Mepp) teste le flag « init » et si sa valeur est 3 réalise alors l'affichage des « connexions » (cf. ci-dessous).

```
void mepp_component_CGAL_Example_plugin::post_draw_all_scene()
{
	Viewer* viewer = (Viewer *)mw->activeMdiChild();
	PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

	if (doesExistComponentForViewer<CGAL_Example_ComponentPtr,
CGAL_Example_Component>(viewer, polyhedron_ptr)) // important !!!
	{
		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);
		if (component_ptr->get_init() == 3) (2)
		{
			int nbMesh = qMin(viewer->getScenePtr()->get_nb_polyhedrons(), viewer-
>get_nb_frames());
			if (nbMesh == 2)
			{
				glPushMatrix();
					draw_connections(viewer, 0, 1); // link between first and
second mesh (first and second frame) (3)
				glPopMatrix();
			}
		}
	}
}
```

Explications :

(2) On teste la valeur du flag « init ».
(3) On dessine les connexions.



3.3.5 Step5 : fixer une position et une orientation pour la « frame » du polyèdre courant (uniquement avec le mode « Space » de Mepp)

Dans mepp_component_CGAL_Example_plugin.cpp :

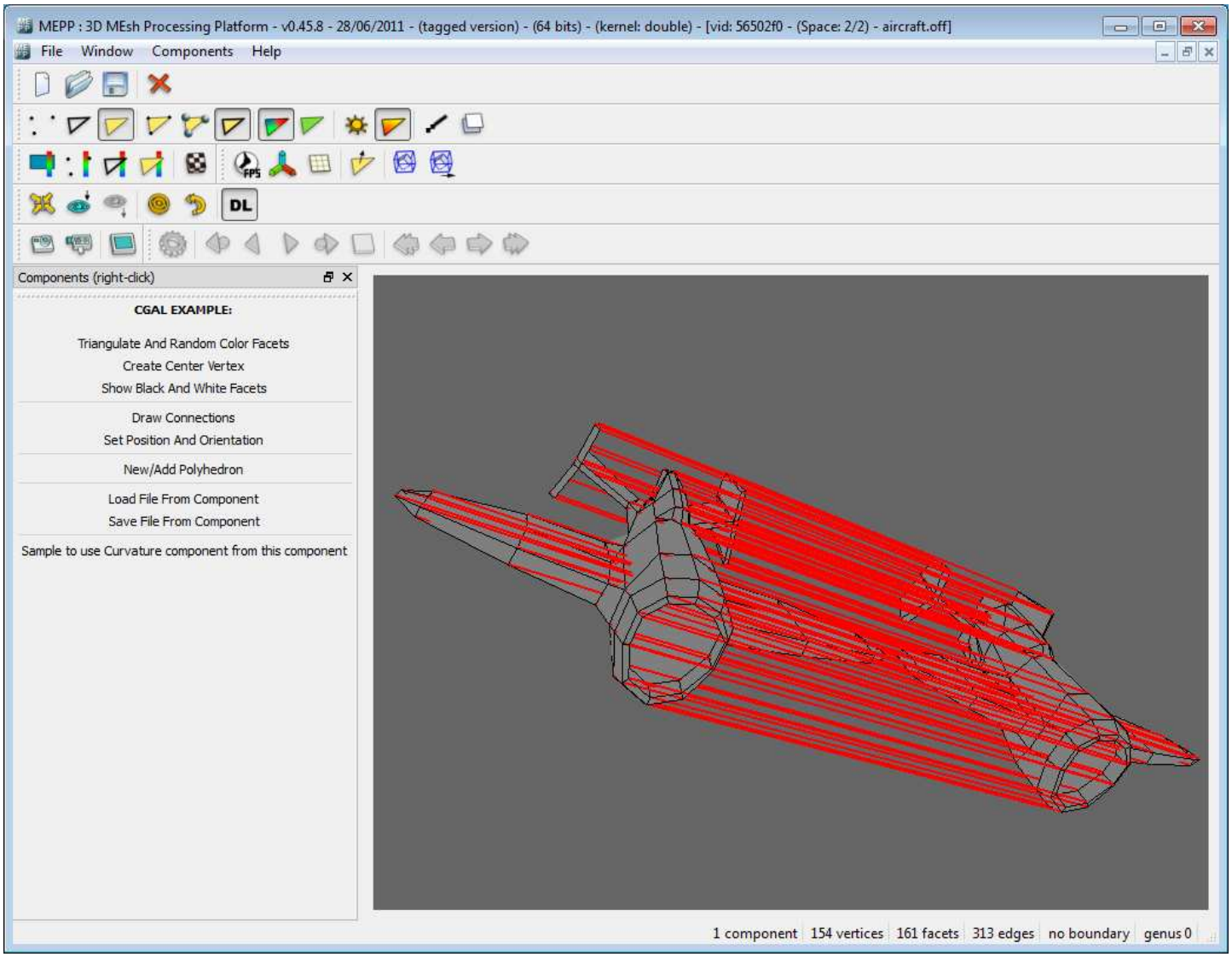
On appelle notre cinquième fonction ci-dessus par le biais de la fonction « step5() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step5()
{
	QApplication::setOverrideCursor(Qt::WaitCursor);

	// active viewer
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);

		Vec pw(2, 0, 0); viewer->frame(viewer->getScenePtr()->get_current_polyhedron())-
>setPosition(pw); // position in world coordinate system (1)

		Quaternion qw(0, 0, 0, 1); // identity quaternion (i.e., no rotation)
		viewer->frame(viewer->getScenePtr()->get_current_polyhedron())->setOrientation(qw); //
rotation in world coordinate system (2)
```

```
            viewer->updateGL(); (3)
        }

        QApplication::restoreOverrideCursor();
}
```

Explications :

(1) On positionne la « frame » du polyèdre courant en (2, 0, 0) dans le repère « monde ».
(2) On oriente la « frame » du polyèdre courant sans aucune rotation sur aucun des axes dans le repère « monde ».
(3) On rafraichit l'affichage.



3.3.6 Step6 : créer une nouvelle « fenêtre fille » avec un « polyèdre exemple » ou rajouter un « polyèdre exemple » à la fenêtre fille courante

Dans mepp_component_CGAL_Example_plugin.cpp :

On appelle notre sixième fonction ci-dessus par le biais de la fonction « step6() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step6()
{
        QApplication::setOverrideCursor(Qt::WaitCursor);
```

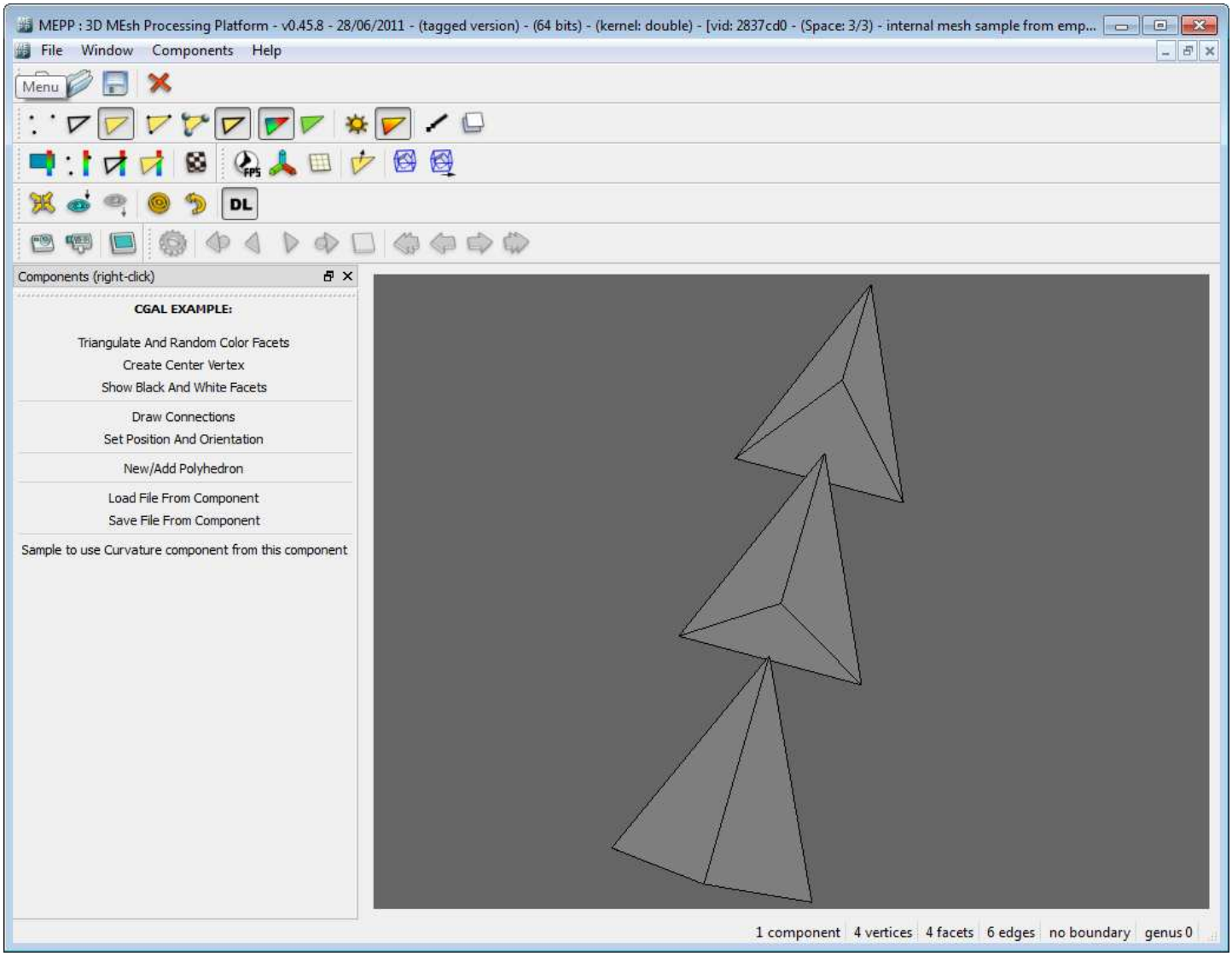
```
	// active viewer
	if (mw->activeMdiChild() != 0) (1)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

		CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);
		// step 6a : begin
		emit(mw->get_actionAddEmpty()->trigger()); (2)

		int nb_polyhedrons = viewer->getScenePtr()->get_nb_polyhedrons(); (3)
		polyhedron_ptr = viewer->getScenePtr()->get_polyhedron(nb_polyhedrons-1);
		component_ptr->CreateTetrahedron(polyhedron_ptr);

		viewer->getScenePtr()->setcurrentFile(tr("internal mesh sample from empty")); (4)
		viewer->setDynTitle();

		viewer->recreateListsAndUpdateGL(); (5)
		// step 6a : end
	}
	else
	{
		// step 6b : begin
		emit(mw->get_actionNewEmpty()->trigger()); (6)

		for (int i=0; i<lwindow.size(); i++) // all viewers
		{
			Viewer* viewer = (Viewer *)qobject_cast<QWidget *>(lwindow[i]->widget());
			if (viewer->getScenePtr()->get_polyhedron()->empty()) (7)
			{
				PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

				CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);
				component_ptr->CreateTetrahedron(polyhedron_ptr); (8)

				viewer->showAllScene(); (9)

				viewer->getScenePtr()->setcurrentFile(tr("internal mesh sample from
empty")); (10)
				viewer->setDynTitle();
			}
		}
		// step 6b : end
	}

	QApplication::restoreOverrideCursor();
}
```

Explications :

(1) S'il y a une « fenêtre fille » ouverte et active dans Mepp :
(2) on émet un signal permettant d'ajouter un polyèdre « vide » au sein de cette « fenêtre fille »,
(3) on récupère ensuite son pointeur et on crée le « polyèdre exemple » : un tétraèdre,
(4) on lui donne un nom et on rafraîchit le titre de la « fenêtre fille »,
(5) on rafraichit l'affichage.

S'il n'y a pas de « fenêtre fille » ouverte et active dans Mepp :
(6) on émet un signal permettant de créer une nouvelle « fenêtre fille »,
(7) on parcourt ensuite toutes les « fenêtres filles » de Mepp et on cherche la « fenêtre fille vide » nouvellement créée,
(8) on récupère ensuite le pointeur du polyèdre « vide » et on crée le « polyèdre exemple » : un tétraèdre,
(9) on centre la « scène » de manière à voir le polyèdre nouvellement créé dans son intégralité,

(10) on lui donne un nom et on rafraîchit le titre de la « fenêtre fille ».



3.3.7 Step7/8 : charger/sauvegarder un polyèdre au sein de son composant (exemple : format de fichier propre au composant)

Dans mepp_component_CGAL_Example_plugin.cpp :

On appelle notre septième fonction ci-dessus par le biais de la fonction « step7() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step7()
{
	emit(mw->get_mainwindowActionOpen()->doSendParamsOpen(tr("Open Mesh File(s) - from CGAL_Example"), tr("OFF files (*.off)"), Normal, mepp_component_plugin_interface::load_file_from_component)); (1)
}
```

Explications :

(1) On émet un signal permettant d'ouvrir une boite de dialogue avec les paramètres « `tr("Open Mesh File(s) - from CGAL_Example"), tr("OFF files (*.off)"), Normal` » et d'invoquer ensuite la fonction « load_file_from_component » propre au composant (cf. ci-dessous).

```
int mepp_component_plugin_interface::load_file_from_component(PolyhedronPtr polyhedron_ptr, QString filename, Viewer* viewer)
```

```
{
	mepp_component_CGAL_Example_plugin *mepp_component_plugin = NULL;
	for (int i=0; i<viewer->lplugin.size(); ++i)
	{
		if (dynamic_cast<mepp_component_CGAL_Example_plugin*>(viewer->lplugin[i]) !=
0)
		{
			mepp_component_plugin =
dynamic_cast<mepp_component_CGAL_Example_plugin*>(viewer->lplugin[i]); (2)
		}
	}

	int res;

	if (mepp_component_plugin)
	{
		CGAL_Example_ComponentPtr component_ptr = mepp_component_plugin-
>findOrCreateComponentForViewer<CGAL_Example_ComponentPtr,
CGAL_Example_Component>(viewer, polyhedron_ptr); (3)

		res = polyhedron_ptr->load_mesh_off(filename.toStdString()); (4)
	}
	else
		res=-1;

	return res;
}
```

Explications :

(2) On récupère un pointeur sur le « plugin » du composant.
(3) On récupère un pointeur sur le composant.
(4) On appelle sa « propre fonction » de chargement de polyèdre (ici, dans cette exemple, il s'agit du chargement par défaut des fichiers « .off » mais il peut s'agir d'un format spécifique (cf. composant Compression_Valence et format P3D).

Note : le principe est le même pour sauvegarder un polyèdre au sein de son composant (mais on utilise à la place la fonction « save_file_from_component » propre au composant).

3.3.8 Step9 : utiliser un autre composant au sein de son composant (mais au préalable cf. points 1.2d et 1.4)

Dans mepp_component_CGAL_Example_plugin.cpp :

On appelle notre neuvième fonction ci-dessus par le biais de la fonction « step9() » qui peut être invoquée par le menu ou la toolbar du composant lui-même.

```
void mepp_component_CGAL_Example_plugin::step9()
{
	// active viewer
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

		QApplication::setOverrideCursor(Qt::WaitCursor);

		// we use CGAL_Example component here (as usual) (1)
			CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);
			component_ptr->TriangulateAndRandomColorFacets(polyhedron_ptr);
```

```
                component_ptr->set_init(2);
            // we use CGAL_Example component here (as usual)

            // we use Curvature component here (2)
                bool IsGeo;
                double radius;
                Curvature_ComponentPtr component_ptr_curvature =
findOrCreateComponentForViewer<Curvature_ComponentPtr, Curvature_Component>(viewer,
polyhedron_ptr);

                // params
                radius = 0.001;
                IsGeo=true;

                mw->statusBar()->showMessage(tr("Curvature..."));
                component_ptr_curvature-
>principal_curvature(polyhedron_ptr,IsGeo,radius);
                mw->statusBar()->showMessage(tr("Curvature is done"));

                component_ptr_curvature->set_init(2);

                component_ptr_curvature->ConstructColorMap(polyhedron_ptr,1);
                viewer->recreateListsAndUpdateGL();
            // we use Curvature component here
        }

        QApplication::restoreOverrideCursor();
}
```

Explications :

(1) On utilise son composant comme d'habitude.
(2) On utilise un autre composant en utilisant un autre pointeur sur le composant en question (mais au préalable cf. **points 1.2d et 1.4**).

3.3.9 Gestion des évènements clavier/souris, interaction avec l'utilisateur

Dans CGAL_Example_Component.h :

Nous allons tout d'abord déclarer la fonction relative à notre étape. Cette fonction sera « publique ».

```
void GetClickedVertices(PolyhedronPtr pMesh, double x, double y, int tolerance);
```

Le corps de cette fonction sera quant à lui présent dans CGAL_Example_Component.cpp :

```
void CGAL_Example_Component::GetClickedVertices(PolyhedronPtr pMesh, double x, double y, int tolerance)
{
        GLdouble *model ;  GLdouble *proj ;  GLint *view;

        view=new int[4096];
        model=new double[4096];
        proj=new double[4096];

        glGetIntegerv (GL_VIEWPORT, view); (1)
        glGetDoublev (GL_MODELVIEW_MATRIX, model);
        glGetDoublev (GL_PROJECTION_MATRIX, proj);

        y=view[3]-y;

        GLdouble wx ; GLdouble wy ; GLdouble wz;

        int vertexID=0;
        for (Vertex_iterator pVertex = pMesh->vertices_begin(); pVertex!= pMesh->vertices_end();
pVertex++)
        {
```

```
			gluProject (pVertex->point().x(),pVertex->point().y(),pVertex->point().z(), model, proj,
view, &wx, &wy, &wz);  // on simule la projection du sommet dans l'espace window (2)
			if (wz>0. && wz<1)
			if (x>floor(wx)-tolerance && x<floor(wx)+tolerance)
			if (y>floor(wy)-tolerance && y<floor(wy)+tolerance)  // on fixe une petite tolérance (de
2*5 pixels par exemple) pour la sélection (3)
			{
				pVertex->color(1., 0., 0.); (4)
			}
			vertexID++;
		}

		delete[]view;
		delete[]model;
		delete[]proj;
}
```

Explications :

(1) On récupère les 3 matrices d'OpenGL (GL_VIEWPORT, GL_MODELVIEW_MATRIX, GL_PROJECTION_MATRIX).
(2) Pour tous les sommets, on projette ses coordonnées dans l'espace « window »,
(3) on fixe une petite tolérance afin de facilité le « clic sur sommet »,
(4) on colorie le sommet (une fois identifié) en rouge.

Dans mepp_component_CGAL_Example_plugin.cpp :

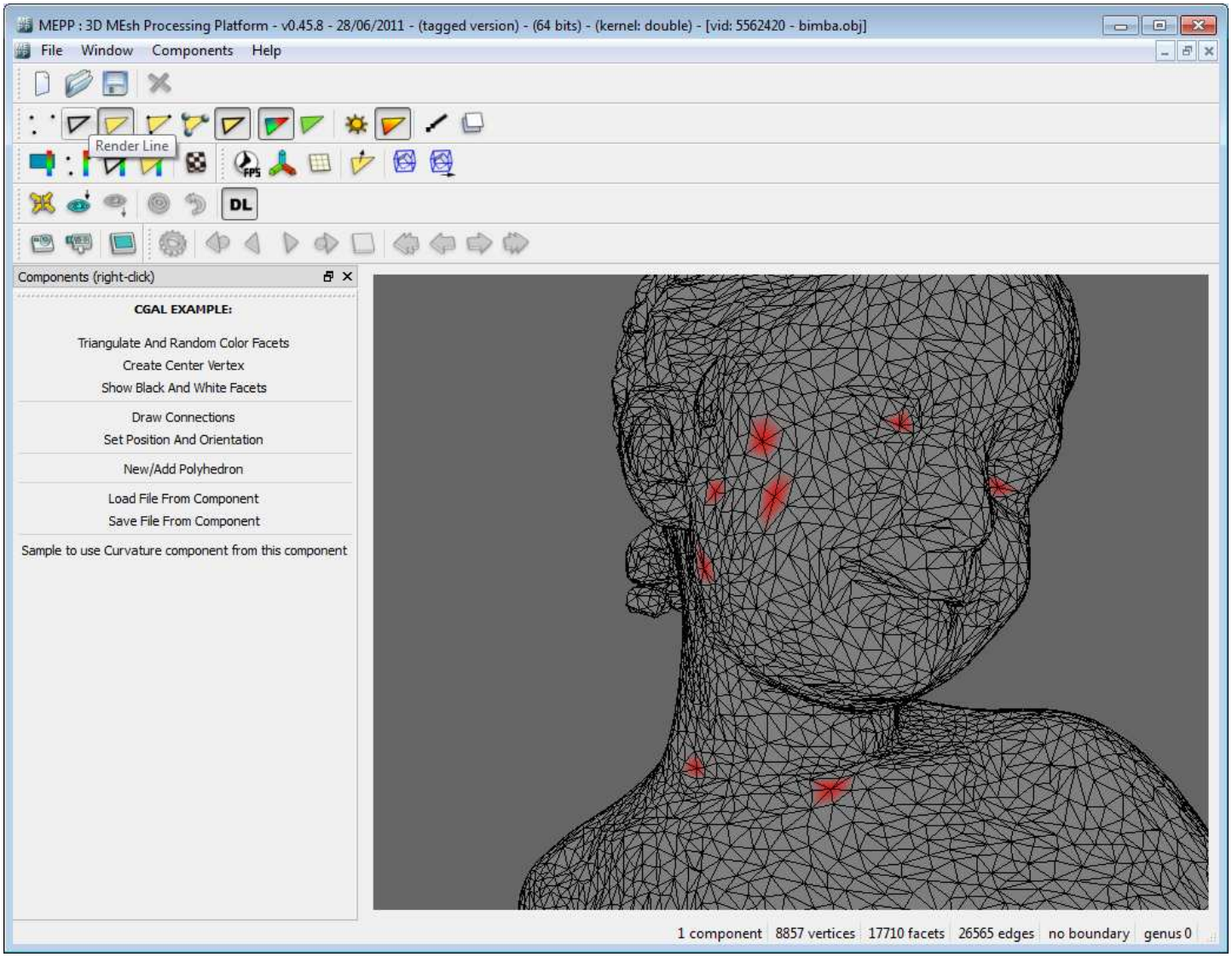
La fonction est appelée par le biais de l'évènement `OnMouseLeftDown(QMouseEvent *event)`.

```
void mepp_component_CGAL_Example_plugin::OnMouseLeftDown(QMouseEvent *event)
{
	if (mw->activeMdiChild() != 0)
	{
		Viewer* viewer = (Viewer *)mw->activeMdiChild();
		PolyhedronPtr polyhedron_ptr = viewer->getScenePtr()->get_polyhedron();

		if (doesExistComponentForViewer<CGAL_Example_ComponentPtr,
CGAL_Example_Component>(viewer, polyhedron_ptr)) // important !!! (5)
		{
			mw->statusBar()->showMessage(tr("mepp_component_CGAL_Example_plugin:
OnMouseLeftDown"), 1000); (6)

			CGAL_Example_ComponentPtr component_ptr =
findOrCreateComponentForViewer<CGAL_Example_ComponentPtr, CGAL_Example_Component>(viewer,
polyhedron_ptr);

			if (viewer->getScenePtr()->get_loadType() == Normal)
			{
				component_ptr->GetClickedVertices(viewer->getScenePtr()->get_polyhedron(),
event->x(), event->y(), 10); (7)
				viewer->recreateListsAndUpdateGL();
			}
		}
	}
}
```

Explications :

(5) On vérifie au préalable (grâce à la fonction « doesExistComponentForViewer ») que ce composant (CGAL_Example) a déjà été instancié au sein de cette « fenêtre fille », en clair qu'il a déjà été invoqué et utilisé. En effet, si ce n'était pas le cas, l'évènement serait alors à ignorer.
Ce test très important permet en fait d'envoyer l'évènement qu'aux composants « actifs », ce sans quoi tous les composants (même « inactifs » recevraient sinon l'évènement).
(6) On renseigne la barre de statut de Mepp afin de signifier qu'un clic vient de se produire et on laisse ce message affiché seulement pendant 1 seconde.

(7) On appelle notre fonction en lui passant notre polyèdre en pointeur ainsi que les coordonnées x, y du « clic écran ». On fixe ici une tolérance de 10 pixels en x et y pour la précision du clic.



4 Rajout de boîtes de dialogue à l'interface d'un composant

4.1 Créer une boîte de dialogue (avec Qt Designer)

a) Localiser Qt Designer :
- sous Windows, cet outil se trouve dans « C:\dev\qt-x.x.x\bin » : designer.exe
- sous Linux, cet outil se trouve dans le menu « Applications > Programmation > Qt 4 Designer »
- sous Mac, cet outil se trouve via « Finder > Applications > MacPorts > Qt4 > Designer »

b) Ouvrir le .ui de votre composant afin de placer vos widgets comme vous le désirez, pour se faire, vous pouvez lire une partie intéressante d'un tutoriel relatif à Qt Designer ici :
http://www.siteduzero.com/tutoriel-3-11360-modeliser-ses-fenetres-avec-qt-designer.html#ss_part_2

Ce dernier présente en effet notamment les notions importantes de « layouts » et « spacers » ainsi que la fenêtre d'édition des propriétés des widgets.

Vous pouvez également ouvrir les .ui des composants suivants afin de vous en inspirer :
- dialSettings.ui du composant Canonical
- dialSettings.ui du composant VSA

- dialSettings*.ui du composant Compression_Valence

Au cours de cette étape il sera surtout IMPORTANT de NOMMER CORRECTEMENT vos widgets à l'aide de l'inspecteur d'objet présent en haut de la colonne de droite de Qt Designer.

c) Sauvegarder votre boîte de dialogue.

4.2 Invoquer votre boîte de dialogue et récupérer les valeurs saisies/renseignées au sein du code de votre composant :

Dans
« trunk\src\components\'**Votre_Catégorie**'\'**Votre_Composant**'\src\mepp_component_Votre_Composant_plugin.cpp », utiliser un code similaire à celui-ci :

```
SettingsDialog_Votre_Composant dial;
if (dial.exec() == QDialog::Accepted)
{
	int iteration = dial.Iteration->value() // 'Iteration' étant un nom de widget (cf. étape ci-dessus)
	...
```

4.3 Note au sujet de l'utilisation de plusieurs boîtes de dialogue au sein de votre composant

a) Dupliquer les 3 fichiers suivants :
- dialSettings_**Votre_Composant**.ui
- dialSettings_**Votre_Composant**.cpp
- dialSettings_**Votre_Composant**.hxx

en

- dialSettings**Fonction_Votre_Composant**.ui
- dialSettings**Fonction_Votre_Composant**.cpp
- dialSettings**Fonction_Votre_Composant**.hxx

('**Fonction**' représentant ici quelque chose en rapport avec cette nouvelle boîte de dialogue.)

b) Ouvrir « dialSettings**Fonction_Votre_Composant**.ui » avec Qt Designer afin de renommer la nouvelle boîte de dialogue à l'aide de l'inspecteur d'objet
(ici la propriété '**objectName**' du QDialog devient « Settings**Fonction** » à la place de « Settings ».)

c) Sauvegarder votre boîte de dialogue.

d) Rechercher et remplacer en respectant la « case » :
- HEADER_MEPP_COMPONENT_**VOTRE_COMPOSANT**_PLUGIN_SETTINGS_H par HEADER_MEPP_COMPONENT_**VOTRE_COMPOSANT**_PLUGIN_SETTINGS_**FONCTION**_H au sein de « dialSettings**Fonction_Votre_Composant**.hxx »
- ui_dialSettings_**Votre_Composant**.h par ui_dialSettings**Fonction_Votre_Composant**.h au sein de « dialSettings**Fonction_Votre_Composant**.hxx »
- Ui_Settings par Ui_Settings**Fonction** au sein de « dialSettings**Fonction_Votre_Composant**.hxx »

- dialSettings_**Votre_Composant**.hxx par dialSettings**Fonction_Votre_Composant**.hxx au sein de « dialSettings**Fonction_Votre_Composant**.cpp »

- SettingsDialog_**Votre_Composant** par SettingsDialog**Fonction_Votre_Composant** au sein des 2 fichiers « dialSettings**Fonction_Votre_Composant**.cpp(.hxx) »

e) Dans
« trunk\src\components\'**Votre_Catégorie**'\'**Votre_Composant**'\src\mepp_component_**Votre_Composant**_plugin.cpp », rajouter sous la ligne 11 la ligne suivante :
#include "dialSettings**Fonction_Votre_Composant**.hxx"

puis utiliser un code similaire à celui-ci pour récupérer les valeurs saisies/renseignées au sein du code de votre composant :

```
SettingsDialog<b>Fonction_Votre_Composant</b> dial<b>Fonction</b>;
if (dial<b>Fonction</b>.exec() == QDialog::Accepted)
{
	int iteration = dial<b>Fonction</b>.Iteration->value()
	...
```