

Source code sample C++

1 Tuio2Mudbox.cpp

This is the main cpp-file of my bachelor thesis' multi touch control plugin.
It handles everything but the multi touch logic.

```
1 // Tuio2Mudbox.cpp
2 // *****/
3 // Based on a sample plug-in by Autodesk, Inc.
4 // Copyright (c) 2008 Autodesk, Inc.
5 // All rights reserved.
6 //
7 // Use of this software is subject to the terms of the Autodesk license
8 // agreement provided at the time of installation or download, or which
9 // otherwise accompanies this software in either electronic or hard copy form.
10 //
11 // *****/
12 // DESCRIPTION:
13 // CREATED: October 2008
14 // *****/
15 // Developed by Marc A. Modrow
16 // Copyright (c) 2010, Marc A. Modrow
17 // This is a Autodesk Mudbox Plugin for input via multi touch interfaces using
18 // the TUIO touch framework.
19
20 #include "Tuio2Mudbox.h"
21 #include "MudboxTuioListener.h"
22
23 #include <QCoreApplication>
24 #include <QApplication>
25 #include <QDesktopWidget>
26 #include <QMouseEvent>
27 #include <QPoint>
28
29 #include "TuioClient.h"
30 #include "TuioCursor.h"
31
32 #define USE_MATH_DEFINES
33 #include <cmath>
34
35 using namespace mudbox;
36
37 // Filter plugin macros
38 IMPLEMENT_VCLASS( Tuio2Mudbox, ViewPortFilter, "Tuio powered multitouch control", 1 );
39 MB_PLUGIN( "Tuio2Mudbox", "Tuio powered multitouch support for Mudbox2011x86", "Marc A. Modrow", "http
://marcam.de.ms", Tuio2Mudbox::Initializer );
40
41 // constants
42 const float DEG2RAD = (float)M_PI/180.0f;
43 const float RAD2DEG = 180.0f/(float)M_PI;
44
45 // globals
46 TUIO::TuioClient *client;
47 MudboxTuioListener *listener;
48
49 // statics
50 //cursorGroups[Hand][Finger]
51 TUIO::TuioCursor* Tuio2Mudbox::CursorGroups[2][5];
52 //average position of each hand
53 Vector* Tuio2Mudbox::handPosition[2];
54 //screen resolutions
55 int Tuio2Mudbox::m_aStatScreenResX(1680);
56 int Tuio2Mudbox::m_aStatScreenResY(1050);
57
58 //offset for tilted view angles
59 float Tuio2Mudbox::m_aStaticXDisplacement(0);
60 //offset for tilted view angles
61 float Tuio2Mudbox::m_aStaticYDisplacement(0);
62 //pixel-AR (PC screens usually have square pixels)
63 float Tuio2Mudbox::aspectRatio(1);
64
65 //velocity sensitivities for:
66 //tool selection
67 float Tuio2Mudbox::m_aStatToolSens(0.1f);
68 //parameter changes
69 float Tuio2Mudbox::m_aStatParamSens(0.02f);
70 //detecting a uniform movement of a cursor group
71 float Tuio2Mudbox::m_aStatUnitySens(0.8f);
72 //max distance for spawning points to count into an existing hand
```

```

73| float Tuio2Mudbox::m_aStatMaxHandDist(0.25f);
74|
75| bool Tuio2Mudbox::redraw(false);
76| bool Tuio2Mudbox::inverted(false);
77|
78| QString Tuio2Mudbox::selectedTool("");
79|
80| void Tuio2Mudbox::Initializer()
81| {
82| }
83|
84| //this defines what cursors belong to the given cursor to form a hand.
85| int Tuio2Mudbox::SortCursorIntoGroup(TUIO::TuioCursor *tcur) {
86|     int firstFreeSlot = -1;
87|     //run for both hands
88|     for (int i = 0; i < 2; i++) {
89|         //meaning: if the cursor [i][0] exists
90|         if (Tuio2Mudbox::CursorGroups[i][0]) {
91|             //max of 5 fingers per hand
92|             for (int j = 0; j < 4; j++) {
93|                 //see above
94|                 if (Tuio2Mudbox::CursorGroups[i][j]) {
95|                     //is the checked cursor closer away from the found one at [i][j]
96|                     //than the threshold for being a hand?
97|                     TUIO::TuioPoint point = Tuio2Mudbox::CursorGroups[i][j]->getPosition();
98|                     //sort given cursor into the first free slot in this hand
99|                     if (tcur->getDistance(&point) < Tuio2Mudbox::m_aStatMaxHandDist) {
100|                         while (j < 5 && Tuio2Mudbox::CursorGroups[i][j]) {
101|                             j++;
102|                         }
103|                         Tuio2Mudbox::CursorGroups[i][j] = tcur;
104|                         return i;
105|                     }
106|                 }
107|             }
108|         }
109|         //if the currently checked slot is not existing it is free
110|         //-- meaning it is the start of a new hand.
111|         else if (firstFreeSlot == -1) {
112|             firstFreeSlot = i;
113|         }
114|     }
115|     //create a new hand if necessary.
116|     if (firstFreeSlot != -1) {
117|         Tuio2Mudbox::CursorGroups[firstFreeSlot][0] = tcur;
118|     }
119|     return firstFreeSlot;
120| }
121|
122| //removes a cursor from a group if no longer needed.
123| //only the cursor is needed, not the group.
124| bool Tuio2Mudbox::DeleteCursorFromGroups(TUIO::TuioCursor *tcur) {
125|     //as there are max. 10 cursors allowed at any given time
126|     //brute forcing is in average as efficient as searching.
127|     for (int i = 0; i < 2; i++) {
128|         if (Tuio2Mudbox::CursorGroups[i][0]) {
129|             for (int j = 0; j < 5; j++) {
130|                 //does [i][j] exist and is it identical to the given cursor?
131|                 //if so delete it and move everything behind it back by one and then return true.
132|                 if (Tuio2Mudbox::CursorGroups[i][j] && Tuio2Mudbox::CursorGroups[i][j]->getSessionID() == tcur
133|                     ->getSessionID()) {
134|                     Tuio2Mudbox::CursorGroups[i][j] = NULL;
135|                     j++;
136|                     while (j < 5) {
137|                         if (Tuio2Mudbox::CursorGroups[i][j]) {
138|                             Tuio2Mudbox::CursorGroups[i][j-1] = Tuio2Mudbox::CursorGroups[i][j];
139|                             Tuio2Mudbox::CursorGroups[i][j] = NULL;
140|                         }
141|                         j++;
142|                     }
143|                     return true;
144|                 }
145|             }
146|         }
147|     }
148|     return false;
149| }
150| //figures out to which hand a cursor belongs.
151| int Tuio2Mudbox::IdentifyCursorGroup (TUIO::TuioCursor *tcur) {
152|     int foundIndex = -1;
153|     //as there are max. 10 cursors allowed at any given time
154|     //brute forcing is in average as efficient as searching.
155|     for (int i = 0; i < 2 && foundIndex == -1; i++) {

```

```

156     if (Tuio2Mudbox::CursorGroups[i][0]) {
157         for (int j = 0; j < 5 && foundIndex == -1 && Tuio2Mudbox::CursorGroups[i][j] ; j++) {
158             if (Tuio2Mudbox::CursorGroups[i][j]->getSessionID() == tcur->getSessionID()) {
159                 foundIndex = i;
160             }
161         }
162     }
163 }
164 return foundIndex;
165 }
166
167 //returns the amount of cursors in a given group.
168 int Tuio2Mudbox::NumberOfCursorsInGroup(int groupIndex) {
169     if (groupIndex != 0 && groupIndex != 1) {
170         return 0;
171     }
172     //checks each element of the group for existence
173     for (int i=4; i>=0; i--) {
174         if (Tuio2Mudbox::CursorGroups[groupIndex][i]) {
175             return i+1;
176         }
177     }
178     return 0;
179 }
180
181 //calculates the average speed of a group
182 mudbox::Vector Tuio2Mudbox::averageGroupSpeeds (int groupIndex) {
183     if (groupIndex != 0 && groupIndex != 1) {
184         return mudbox::Vector(0,0,0);
185     }
186     float velX = 0;
187     float velY = 0;
188     int items = 0;
189     for (items; items < 5 && Tuio2Mudbox::CursorGroups[groupIndex][items]; items++) {
190         velX += Tuio2Mudbox::CursorGroups[groupIndex][items]->getXSpeed();
191         velY += Tuio2Mudbox::CursorGroups[groupIndex][items]->getYSpeed();
192     }
193     velX /= (float)items;
194     velY /= (float)items;
195     return mudbox::Vector(velX, velY);
196 }
197
198 //decide whether a movement of a cursor is unified with its group or individual
199 bool Tuio2Mudbox::isMovementUnified (TUIO::TuioCursor* tcur) {
200     //get its group
201     int groupIndex = Tuio2Mudbox::IdentifyCursorGroup(tcur);
202     if (groupIndex != 0 && groupIndex != 1) {
203         return false;
204     }
205     bool movementUnified = true;
206     TUIO::TuioCursor* tcur0 = Tuio2Mudbox::CursorGroups[groupIndex][0];
207     for (int i = 0; movementUnified && Tuio2Mudbox::CursorGroups[groupIndex][i]; i++) {
208         TUIO::TuioCursor* tcurI = Tuio2Mudbox::CursorGroups[groupIndex][i];
209         //compare the movement difference to the uniformity sensitivity value
210         if (fabs(tcur0->getXSpeed() - tcurI->getXSpeed()) > Tuio2Mudbox::m_aStatUnitySens || fabs(tcur0->
                getYSpeed() - tcurI->getYSpeed()) > Tuio2Mudbox::m_aStatUnitySens) {
211             movementUnified = false;
212         }
213     }
214     return movementUnified;
215 }
216
217 //define where the group has its center.
218 //here it is defined as mathematical average of all coordinates.
219 mudbox::Vector Tuio2Mudbox::getCenterOfGroup(int groupIndex) {
220     if (groupIndex != 0 && groupIndex != 1) {
221         return mudbox::Vector(0,0,0);
222     }
223     float centerX = 0;
224     float centerY = 0;
225     int items = 0;
226     for (items; items < 5 && Tuio2Mudbox::CursorGroups[groupIndex][items]; items++) {
227         centerX += Tuio2Mudbox::CursorGroups[groupIndex][items]->getPosition().getX();
228         centerY += Tuio2Mudbox::CursorGroups[groupIndex][items]->getPosition().getY();
229     }
230     centerX /= (float)items;
231     centerY /= (float)items;
232     return mudbox::Vector(centerX, centerY);
233 }
234
235 //calculates the angle in degrees of the connecting lines from a source point to two aimed points.
236 float Tuio2Mudbox::getAngle(TUIO::TuioPoint *source, TUIO::TuioPoint *aim1, TUIO::TuioPoint *aim2) {
237     //distances:
238     float a = source->getDistance(aim1);

```

```

239 float b = source->getDistance(aim2);
240 float c = aim1->getDistance(aim2);
241 //law of cosines to get the angle's cosines
242 float alpha = acos((b*b + c*c - a*a)/(2*b*c))*RAD2DEG;
243 return alpha;
244 }
245
246 //draws a circle for the feedback on the screen
247 static void drawCircle(float displacementX, float displacementY) {
248     //preps
249     glMatrixMode( GL_PROJECTION );
250     glLoadIdentity();
251     glMatrixMode( GL_MODELVIEW );
252     glLoadIdentity();
253
254     //creating filled circle
255     glBegin( GL_TRIANGLE_FAN );
256     //circle radius
257     float radius = 0.03f;
258     //yellow with 50% opacity
259     glTexCoord4f( 1, 1, 0, 0.5f );
260     //draw a 36 segment circle - approximation, 10 degrees each.
261     for (int i=0; i < 360; i+=10){
262         float degInRad = i*DEG2RAD;
263         glVertex2f(((cos(degInRad)*radius)*Tuio2Mudbox::aspectRatio)+displacementX, (sin(degInRad)*radius)
264             +displacementY);
265     }
266     glEnd();//finishing filled circle
267
268     //black border around the circle.
269     glBegin( GL_LINE_LOOP );
270     //opaque black
271     glTexCoord4f( 0.0f, 0.0f, 0.0f, 1);
272     for (int i=0; i < 360; i+=10){
273         float degInRad = i*DEG2RAD;
274         glVertex2f(((cos(degInRad)*radius)*Tuio2Mudbox::aspectRatio)+displacementX, (sin(degInRad)*radius)
275             +displacementY);
276     }
277     glEnd();//finish
278
279     //crosshair dots, still black
280     glBegin( GL_POINTS );
281     glVertex2f(displacementX, displacementY);
282     glVertex2f(displacementX + radius/3*Tuio2Mudbox::aspectRatio, displacementY);
283     glVertex2f(displacementX - radius/3*Tuio2Mudbox::aspectRatio, displacementY);
284     glVertex2f(displacementX + radius/6*Tuio2Mudbox::aspectRatio, displacementY);
285     glVertex2f(displacementX - radius/6*Tuio2Mudbox::aspectRatio, displacementY);
286     glVertex2f(displacementX, displacementY + radius/3);
287     glVertex2f(displacementX, displacementY - radius/3);
288     glVertex2f(displacementX, displacementY + radius/6);
289     glVertex2f(displacementX, displacementY - radius/6);
290     glEnd();//finish
291 }
292
293 //draw triangle to mark a circle as part of one of the groups
294 static void drawTriangle(mudbox::Vector pos, int groupIndex) {
295     glMatrixMode( GL_PROJECTION );
296     glLoadIdentity();
297     glMatrixMode( GL_MODELVIEW );
298     glLoadIdentity();
299     float radius = 0.015f;
300     //draw triangle group 0: green, group 1:red
301     glTexCoord4f( groupIndex, 1-groupIndex, 0, 0.5f);
302     glBegin( GL_TRIANGLES );
303     for (int i=90; i < 450; i+=120){
304         float degInRad = i*DEG2RAD;
305         glVertex2f(((cos(degInRad)*radius)*Tuio2Mudbox::aspectRatio)+pos.x, (sin(degInRad)*radius)+pos.y);
306     }
307     glEnd();
308
309     //black border again
310     glTexCoord3f( 0, 0, 0);
311     glBegin( GL_LINE_LOOP );
312     for (int i=90; i < 450; i+=120){
313         float degInRad = i*DEG2RAD;
314         glVertex2f(((cos(degInRad)*radius)*Tuio2Mudbox::aspectRatio)+pos.x, (sin(degInRad)*radius)+pos.y);
315     }
316     glEnd();
317
318     glBegin( GL_POINTS );
319     glVertex2f(pos.x, pos.y);
320     glEnd();
321 }

```

```

321 //draw an enclosing outline that c
322 static void drawGroupOutlines(){
323     glMatrixMode( GL_PROJECTION );
324     glLoadIdentity();
325     glMatrixMode( GL_MODELVIEW );
326     glLoadIdentity();
327     //for each hand
328     for (int i = 0; i < 2; i++) {
329         //ich it has a first cursor aka it exists
330         if (Tuio2Mudbox::CursorGroups[i][0]) {
331             glTexCoord3f( i, 1-i, 0);
332             //draw a coloured line
333             glBegin( GL_LINE_LOOP );
334             for (int j = 0; j < 5 && Tuio2Mudbox::CursorGroups[i][j]; j++) {
335                 //get each cursor's position and connect
336                 TUIO::TuioPoint centerPoint(Tuio2Mudbox::CursorGroups[i][j]->getPosition());
337                 //translation from screen coordinates (from touch) to world/window coordinates
338                 QPoint glob(centerPoint.getScreenX(Tuio2Mudbox::m_aStatScreenResX), centerPoint.getScreenY(
339                     Tuio2Mudbox::m_aStatScreenResY));
340                 QWidget *widget = Kernel()->MainWindow()->childAt(glob);
341                 if (widget && (Kernel()->Viewport()->Height() == widget->height() && Kernel()->Viewport()->
342                     Width() == widget->width()))
343                 {
344                     QPoint pos(widget->mapFromGlobal(glob));
345                     float x = pos.x() / (float)widget->width() * 2.0f - 1.0f;
346                     float y = 1.0f - pos.y() / (float)widget->height() * 2.0f;
347                     glVertex2f(x, y);
348                 }
349             }
350             glEnd();
351         }
352     }
353 }
354 //redraw command for all markers and other feedback
355 static void gpuTransform(Texture* const outputTexture) {
356     outputTexture->SetAsRenderTarget();
357     std::list<TUIO::TuioCursor*> cursors = client->getTuioCursors();
358     //cycle through all cursors
359     for (std::list<TUIO::TuioCursor*>::const_iterator it = cursors.begin(); it != cursors.end(); ++it){
360         QPoint glob((*it)->getPosition().getScreenX(Tuio2Mudbox::m_aStatScreenResX), (*it)->getPosition().
361             getScreenY(Tuio2Mudbox::m_aStatScreenResY));
362         QWidget *widget = Kernel()->MainWindow()->childAt(glob);
363         if (widget)
364         {
365             if (Kernel()->Viewport()->Height() == widget->height() && Kernel()->Viewport()->Width() ==
366                 widget->width())
367             {
368                 //get render-target-coordinate
369                 QPoint pos(widget->mapFromGlobal(glob));
370                 float x = pos.x() / (float)widget->width() * 2.0f - 1.0f;
371                 float y = 1.0f - pos.y() / (float)widget->height() * 2.0f;
372                 //draw the marker for that finger
373                 drawCircle(x, y);
374             }
375             else
376             {
377                 Kernel()->Log("warning: cursor out of viewport in gpuTransform()\n");
378             }
379         }
380         else
381         {
382             Kernel()->Log("warning: no widget for cursor in gpuTransform()\n");
383         }
384     }
385 }
386 drawGroupOutlines();
387 //cycle through hands
388 for (int i = 0; i < 2; i++) {
389     if (Tuio2Mudbox::CursorGroups[i][0]) {
390         mudbox::Vector groupCenter = Tuio2Mudbox::getCenterOfGroup(i);
391         TUIO::TuioPoint loc = TUIO::TuioPoint(groupCenter.x, groupCenter.y);
392         QPoint glob(loc.getScreenX(Tuio2Mudbox::m_aStatScreenResX), loc.getScreenY(Tuio2Mudbox::
393             m_aStatScreenResY));
394         QWidget *widget = Kernel()->MainWindow()->childAt(glob);
395         if (widget)
396         {
397             if (Kernel()->Viewport()->Height() == widget->height() && Kernel()->Viewport()->Width() ==
398                 widget->width())
399             {
400                 QPoint pos(widget->mapFromGlobal(glob));
401                 float x = pos.x() / (float)widget->width() * 2.0f - 1.0f;
402                 float y = 1.0f - pos.y() / (float)widget->height() * 2.0f;
403                 mudbox::Vector center(x, y);
404                 //draw a colour coded triangle in the center of that hand

```

```

400         drawTriangle(center , i);
401     }
402     else
403     {
404         Kernel()->Log("warning: coords out of viewport for triangle %i in gpuTransform()\n", i);
405     }
406 }
407 else
408 {
409     Kernel()->Log("warning: no widget for triangle %i in gpuTransform()\n", i);
410 }
411 }
412 }
413 outputTexture->RestoreRenderTarget();
414 }
415
416 //if the mudbox window gets moved or resized refresh all static values and redraw
417 void Tuio2Mudbox::OnNodeEvent( const Attribute &cAttribute , NodeEventType eType )
418 {
419     if(eType == etValueChanged)
420     {
421         Tuio2Mudbox::m_aStatScreenResX = Tuio2Mudbox::m_aScreenResX;
422         Tuio2Mudbox::m_aStatScreenResY = Tuio2Mudbox::m_aScreenResY;
423         Tuio2Mudbox::m_aStatToolSens = Tuio2Mudbox::m_aToolSens;
424         Tuio2Mudbox::m_aStatParamSens = Tuio2Mudbox::m_aParamSens;
425         Tuio2Mudbox::m_aStatUnitySens = Tuio2Mudbox::m_aUnitySens;
426         Tuio2Mudbox::m_aStatMaxHandDist = Tuio2Mudbox::m_aMaxHandDist;
427         Kernel()->Redraw();
428     }
429 }
430
431 //constructor
432 Tuio2Mudbox::Tuio2Mudbox() :
433     //used by the Mudbox plugin administration
434     m_aOverlay(this, "Overlay"),
435     m_aScreenResX(this, "Screen Resolution X"),
436     m_aScreenResY(this, "Screen Resolution Y"),
437     m_aParamSens(this, "Parameter change sensitivity"),
438     m_aToolSens(this, "Brush change sensitivity"),
439     m_aUnitySens(this, "Unity of movement"),
440     m_aMaxHandDist(this, "Hand size")
441 {
442     Kernel()->Log("\n\n\n");
443     // Cg context
444     m_CGContext = cgCreateContext();
445     cgGLSetDebugMode( CG.FALSE );
446
447     cgGLSetManageTextureParameters(m_CGContext, CG.TRUE);
448     cgSetParameterSettingMode(m_CGContext, CG.DEFERRED_PARAMETER_SETTING);
449
450     /* Compile and load the vertex program. */
451     m_FragmentProfile = cgGLGetLatestProfile(CG.GL_FRAGMENT);
452     cgGLSetOptimalOptions(m_FragmentProfile);
453
454     //set the important global values
455     m_aOverlay = true;
456     m_a_prevOverlay = true;
457     inverted = false;
458     m_aParamSens = 0.02f;
459     m_aToolSens = 0.1f;
460     m_aUnitySens = 0.6f;
461     m_aMaxHandDist = 0.25f;
462     QRect desktop = QApplication::desktop()->screenGeometry();
463     m_aScreenResX = desktop.width();// 1680;
464     m_aScreenResY = desktop.height();// 1050;
465     m_aStatScreenResX = m_aScreenResX;
466     m_aStatScreenResY = m_aScreenResY;
467
468     //TUIO essentials
469     listener = new MudboxTuioListener;
470     client = new TUIO::TuioClient;
471     client->addTuioListener(listener); // registers the TuioListener
472     client->connect();
473
474     // Create the output texture
475     m_pResultTexture = CreateInstance<Texture>();
476 }
477
478 //destructor
479 //closes connection of client and tracker and terminates the client side
480 Tuio2Mudbox::~Tuio2Mudbox()
481 {
482     if( m_pResultTexture )
483         delete m_pResultTexture;

```

```

484     if (m.CGContext)
485         cgDestroyContext(m.CGContext);
486     client->disconnect();
487     delete client;
488     delete listener;
489 };
490
491 //rendering
492 void Tuio2Mudbox::Process( ViewPortState &s )
493 {
494     try
495     {
496         glEnable(GL_BLEND);
497         glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
498         enum Image::Format eFormat = s.m.bHDRNeeded ? Image::e16float : Image::e8integer;
499
500         // Ensure our output texture is same size & type as the incoming color buffer.
501         // Get width and height of the viewport
502         const int width = s.m.pColor->Width();
503         const int height = s.m.pColor->Height();
504         // Smallest of width and height
505         const int extendMin = width < height ? width : height;
506
507         // Logarithm of the SMALLER extend, in order to be at least 1 pixel in size in any dimension
508         const int levelCountMax = (int)floorf(logf(float(extendMin)) / float(MLN2));
509
510         Tuio2Mudbox::aspectRatio = ((float) height) / ((float) width);
511
512         // See if the input size or format changed. If so, re-create the texture.
513         if(m.pResultTexture->Width() != s.m.pColor->Width() ||
514            m.pResultTexture->Height() != s.m.pColor->Height() || redraw)
515         {
516             m.pResultTexture->Create(s.m.pColor->Width(), s.m.pColor->Height(), 4, eFormat);
517             redraw = false;
518         }
519         s.m.pColor->CopyTo(*m.pResultTexture);
520         m.pResultTexture->SetLocation( TexturePool::locationGPU );
521
522         // Enable Cg fragment profile
523         cgGLEnableProfile(m.FragmentProfile);
524
525         // Render to the resulting texture. This activates the shader pass.
526         if (Tuio2Mudbox::m.aOverlay)
527         {
528             try {
529                 //writing directly to colour buffer causes instability, but grants visible scene instead of
530                 //black background
531                 gpuTransform(m.pResultTexture/*s.m.pColor*/);
532             } catch (...) {
533                 Kernel()->Log("warning: System exception in gpuTransform().\n");
534             }
535         }
536
537         cgGLDisableProfile(m.FragmentProfile);
538
539         // Assign the resulting texture to the color buffer
540         m.pResultTexture->CopyTo(*s.m.pColor);
541     }
542     catch (...) {
543         Kernel()->Log("warning: System exception in precess().\n");
544     }
545 };
546
547 //toggles if the feedback (circles, triangles, etc.) is visible or not.
548 void Tuio2Mudbox::SetVisible( bool bVisible )
549 {
550     ViewPortFilter::SetVisible( bVisible );
551
552     if( bVisible )
553     {
554         if( !m.pResultTexture )
555             m.pResultTexture = CreateInstance<Texture>();
556     }
557     else
558     {
559         delete m.pResultTexture;
560         m.pResultTexture = 0;
561     }
562 };

```

./c++/Tuio2Mudbox.cpp

2 MudboxTuioListener.cpp

From the same project this file handles all the touch logic and its consequences.

```
1 // Passages replaced with //[...] are lengthy, repetitive or otherwise not interesting.
2 // Where necessary a comment is given in the [] to explain what happens there
3 // MudboxTuioListener
4 //*****
5 // Created upon a sample by tuio.org
6 // Developed by Marc A. Modrow
7 // Copyright (c) 2010, Marc A. Modrow
8 // This is the listener for a Mudbox multi touch control plugin working with TUIO.
9 //
10 // Due to limitations of the Mudbox API the multitouch gestures cannot interact
11 // with the tools directly, but are mapped to mouse and keyboard commands.
12 //*****
13
14 // own stuff
15 #include "MudboxTuioListener.h"
16 #include "Tuio2Mudbox.h"
17
18 // qt stuff
19 #include <QCoreApplication>
20 #include <QMouseEvent>
21 #include <QKeyEvent>
22 #include <QPoint>
23 #include <QWidget>
24
25 #include "Mudbox/mudbox.h"
26
27 using namespace mudbox;
28
29 bool mouse1IsPressed(false);
30 bool mouse2IsPressed(false);
31 float lastStrength(0);
32 long delay(0);
33 bool altPressed(false);
34
35 //constructor and destructor are necessary, but have nothing to do.
36 MudboxTuioListener::MudboxTuioListener()
37 {
38 }
39 MudboxTuioListener::~MudboxTuioListener()
40 {
41 }
42
43 //used to select tools
44 //due to limitations of the Mudbox API the tools cannot be used directly, but have to be
45 //used through a mouse & keyboard simulation
46 void MudboxTuioListener::selectTool(QString name) {
47     //new brush is selected
48     if (name != Tuio2Mudbox::selectedTool){
49         try
50         {
51             //get data for center of operation for the tool
52             mudbox::Vector groupCenter = Tuio2Mudbox::getCenterOfGroup(0);
53             TUIO::TuioPoint loc = TUIO::TuioPoint(groupCenter.x, groupCenter.y);
54             QPoint glob(loc.getScreenX(Tuio2Mudbox::m_aStatScreenResX), loc.getScreenY(Tuio2Mudbox::
55                 m_aStatScreenResY));
56             Tuio2Mudbox::selectedTool = name;
57             if (name != "Not Mapped!") {
58                 if (triggerMouseReleaseButton(glob.x(), glob.y(), Qt::LeftButton, Qt::LeftButton) )
59                 {
60                     if (triggerMouseMoveEvent(glob.x(), glob.y()))
61                     {
62                         Kernel()->SetStatus(Kernel::stNormal, "Switched to " + Tuio2Mudbox::selectedTool);
63                         if (Tuio2Mudbox::selectedTool == "Sculpt")
64                         {
65                             triggerKeyPressEvent(Qt::Key_1);
66                             triggerKeyReleaseEvent(Qt::Key_1);
67                         }
68                     }
69                     // [...] likewise select the tool whose name is given]
70                 } else {
71                     Kernel()->Log("%i error in selectTool(" + name + "). " + name + "is no valid identifier!\n",
72                         Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().getSeconds());
73                 }
74             } //if no mapping is used, button is released
75         } else {
76             //release mouse button to end previous stroke
77             if (triggerMouseReleaseButton(glob.x(), glob.y(), Qt::LeftButton, Qt::LeftButton) )
```



```

78     {
79         Kernel()->Log("%i Mouse released at X:%i Y:%i in selectTool(" + name + ") due to unmapped
            action\n", Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().getSeconds(),
            glob.x(), glob.y());
80     }
81     else {
82         Kernel()->Log("%i error while releasing Mouse at X:%i Y:%i in selectTool(" + name + ") due
            to unmapped action\n", Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().
            getSeconds(), glob.x(), glob.y());
83     }
84     Kernel()->SetStatus(Kernel::stNormal, "No Brush mapped to this action.");
85 }
86 }
87 catch (...)
88 {
89     Kernel()->Log("warning: System exception in selectTool("+name+").\n");
90 }
91 }
92 }
93 }
94 //creates key press events
95 bool MudboxTuioListener::triggerKeyPressEvent(int button, int modifiers)
96 {
97     QPoint cursor = QCursor::pos();
98     QWidget *receiver = Kernel()->MainWindow()->childAt(cursor);
99     QKeyEvent *keyEvent = new QKeyEvent(QEvent::KeyPress, button, (Qt::KeyboardModifiers)modifiers,
        QString(button));
100    QApplication::postEvent(receiver, keyEvent);
101    return true;
102 }
103 }
104 //creates key release events
105 bool MudboxTuioListener::triggerKeyReleaseEvent(int button, int modifiers)
106 {
107     QPoint cursor = QCursor::pos();
108     QWidget *receiver = Kernel()->MainWindow()->childAt(cursor);
109     QKeyEvent *keyEvent = new QKeyEvent(QEvent::KeyRelease, button, (Qt::KeyboardModifiers)modifiers,
        QString(button));
110    QApplication::postEvent(receiver, keyEvent);
111    return true;
112 }
113 }
114 //creates mouse move events
115 bool MudboxTuioListener::triggerMouseMoveEvent(int x, int y, int button, int buttons)
116 {
117     QWidget *receiver = Kernel()->MainWindow()->childAt(x, y);
118     //only execute when the mouse would move to a legal place
119     if (receiver)
120     {
121         long now(0);
122         if (Tuio2Mudbox::CursorGroups[0][0]) {
123             //gets the age of cursor [0][0]
124             now = Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().getTotalMilliseconds();
125             //check if the cursor is older than 250ms, a brush is selected and it's strength is
126             //0. If so restore the last used brush strength.
127             if (now - delay > 250 && Kernel()->m_pSelectedBrush && Kernel()->m_pSelectedBrush->BrushStrength
                () == 0)
128             {
129                 Kernel()->m_pSelectedBrush->SetBrushStrength(lastStrength);
130                 delay = 0;
131             }
132         }
133         if (mouse1IsPressed)
134         {
135             button = Qt::LeftButton;
136             buttons = Qt::LeftButton;
137         }
138         else if (mouse2IsPressed)
139         {
140             button = Qt::MidButton;
141             buttons = Qt::MidButton;
142         }
143         else
144         {
145             button = 0;
146             buttons = 0;
147         }
148         QPoint globPos(x, y);
149         QPoint locPos(receiver->mapFromGlobal(globPos));
150         QMouseEvent *mouseEvent = new QMouseEvent(QEvent::MouseMove, locPos, globPos, (Qt::MouseButton)
            button, (Qt::MouseButtons)buttons, 0);
151         QApplication::postEvent(receiver, mouseEvent);
152         return true;
153     }
154     return false;

```

```

155 }
156
157 //creates mouse button press events
158 bool MudboxTuioListener::triggerMousePressButton(int x, int y, int button, int buttons, int
    keyboardModifier)
159 {
160
161     if (keyboardModifier == 0 && Tuio2Mudbox::inverted) {
162         keyboardModifier = Qt::ControlButton;
163     }
164     QWidget *receiver = Kernel()->MainWindow()->childAt(x, y);
165     if (receiver)
166     {
167         //when a new stroke is being started
168         if (Kernel()->m_pSelectedBrush && Kernel()->m_pSelectedBrush->BrushStrength() != 0 && !(receiver->
            height() != Kernel()->ViewPort()->Height() || receiver->width() != Kernel()->ViewPort()->Width
            ()))
169         {
170             lastStrength = Kernel()->m_pSelectedBrush->BrushStrength();
171             Kernel()->m_pSelectedBrush->SetBrushStrength(0);
172             //wait 0.3 sec;
173             if (Tuio2Mudbox::CursorGroups[0][0]) {
174                 delay = Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().getTotalMilliseconds();
175             } else {
176                 delay = 0;
177             }
178         }
179         if (button == Qt::LeftButton)
180             mouse1IsPressed = true;
181         if (buttons == Qt::MidButton)
182             mouse2IsPressed = true;
183         QPoint globPos(x, y);
184         QPoint locPos(receiver->mapFromGlobal(globPos));
185         QMouseEvent *mouseEvent = new QMouseEvent(QEvent::MouseButtonPress, locPos, globPos, (Qt::
            MouseButton)button, (Qt::MouseButtons)buttons, (Qt::KeyboardModifier)keyboardModifier);
186         QCoreApplication::postEvent(receiver, mouseEvent);
187         return true;
188     }
189     return false;
190 }
191
192 //creates mouse button release events
193 bool MudboxTuioListener::triggerMouseReleaseButton(int x, int y, int button, int buttons, int
    keyboardModifier)
194 {
195     if (keyboardModifier == 0 && Tuio2Mudbox::inverted) {
196         keyboardModifier = Qt::ControlButton;
197     }
198     QWidget *receiver = Kernel()->MainWindow()->childAt(x, y);
199     if (receiver)
200     {
201         mouse1IsPressed = false;
202         mouse2IsPressed = false;
203         QPoint globPos(x, y);
204         QPoint locPos(receiver->mapFromGlobal(globPos));
205         QMouseEvent *mouseEvent = new QMouseEvent(QEvent::MouseButtonRelease, locPos, globPos, (Qt::
            MouseButton)button, (Qt::MouseButtons)buttons, (Qt::KeyboardModifier)keyboardModifier);
206         QCoreApplication::postEvent(receiver, mouseEvent);
207         if (button != buttons)
208         {
209             QMouseEvent *mouseEvent = new QMouseEvent(QEvent::MouseButtonRelease, locPos, globPos, (Qt::
                MouseButton)buttons, (Qt::MouseButtons)buttons, (Qt::KeyboardModifier)keyboardModifier);
210             QCoreApplication::postEvent(receiver, mouseEvent);
211         }
212         return true;
213     }
214     return false;
215 }
216
217 //creates a new cursor when a new finger is placed on the screen
218 void MudboxTuioListener::addTuioCursor(TUIO::TuioCursor *tcur)
219 {
220     int groupIndex = -1;
221     try
222     {
223         //tapped into viewport?
224         QPoint cursor = QPoint(tcur->getPosition().getScreenX(Tuio2Mudbox::m_aStatScreenResX), tcur->
            getPosition().getScreenY(Tuio2Mudbox::m_aStatScreenResY));
225         QWidget *receiver = Kernel()->MainWindow()->childAt(cursor);
226         if (receiver->height() != Kernel()->ViewPort()->Height() || receiver->width() != Kernel()->
            ViewPort()->Width())
227         {
228             if (triggerMouseMoveEvent(cursor.x(), cursor.y()) )
229             {
230                 triggerMousePressButton(cursor.x(), cursor.y(), Qt::LeftButton, Qt::LeftButton);

```

```

231     Kernel()->Log("%i Mouse pressed at X:%i Y:%i in addTuioCursor() due to click outside of
        viewport.\n", Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().getSeconds(),
        cursor.x(), cursor.y());
232     }
233 }
234 //only cursors in the viewport are added to hands
235 else {
236     groupIndex = Tuio2Mudbox::SortCursorIntoGroup(tcur);
237 }
238 }
239 catch (...) {
240     Kernel()->Log("warning: System exception in addTuioCursor() at click outside of viewport.\n");
241 }
242 int numberOfCursors;
243 if (groupIndex == 0 || groupIndex == 1) {
244     numberOfCursors = Tuio2Mudbox::NumberOfCursorsInGroup (groupIndex);
245 }
246 if(groupIndex == 0) { //strong hand
247     switch(numberOfCursors) {
248         case 1 : MudboxTuioListener::selectTool(" Knife");
249             break;
250         case 2 : MudboxTuioListener::selectTool(" Sculpt");
251             break;
252         case 3 :
253             try {
254                 //get angle between the cursors of this hand to identify aligned cursors.
255                 TUIO::TuioPoint tpoint1 = Tuio2Mudbox::CursorGroups[groupIndex][1]->getPosition();
256                 TUIO::TuioPoint tpoint2 = Tuio2Mudbox::CursorGroups[groupIndex][2]->getPosition();
257                 float alpha1 = Tuio2Mudbox::CursorGroups[groupIndex][0]->getAngleDegrees(&tpoint1);
258                 float alpha2 = Tuio2Mudbox::CursorGroups[groupIndex][0]->getAngleDegrees(&tpoint2);
259                 float angle = fabs(alpha1 - alpha2);
260                 if ((fabs(angle) > 170 && fabs(angle) < 190 ) || fabs(angle) < 7 || fabs(angle) > 350) {
261                     MudboxTuioListener::selectTool(" Eraser");
262                     //select eraser;
263                 } else if (Tuio2Mudbox::selectedTool != "Foamy" && Tuio2Mudbox::selectedTool != "Grab"){
264                     MudboxTuioListener::selectTool(" Flatten");
265                     //deselect eraser;
266                 }
267             }
268             catch (...)
269             {
270                 Kernel()->Log("warning: System exception in addTuioCursor() while handling 1st hand 3 finger
                event.\n");
271             }
272             break;
273         case 4: MudboxTuioListener::selectTool(" Grab");
274             break;
275         case 5: MudboxTuioListener::selectTool(" Not Mapped!");
276             break;
277     }
278 }
279 else if(groupIndex == 1) { //weak hand - special mappings
280     switch(numberOfCursors) {
281         case 4 :
282             try {
283                 //4 finger adding for inverting brush by pressing/releasing the control key.
284                 Tuio2Mudbox::inverted = !Tuio2Mudbox::inverted;
285                 Tuio2Mudbox::inverted ? triggerKeyPressEvent(Qt::Key_Control) : triggerKeyReleaseEvent(Qt::
                Key_Control) ;
286                 Kernel::Kernel().SetStatus(Kernel::stNormal, Tuio2Mudbox::inverted ? "brush inverted" : "
                brush de-inverted");
287             }
288             catch (...)
289             {
290                 Kernel()->Log("warning: System exception in addTuioCursor() while handling 2nd hand 4 finger
                event.\n");
291             }
292             break;
293         case 5:
294             altPressed = true;
295             break;
296     }
297 }
298
299 Tuio2Mudbox::redraw = true;
300 Kernel()->Redraw();
301 }
302
303 //removes a cursor when the finger is lifted off the screen
304 void MudboxTuioListener::removeTuioCursor(TUIO::TuioCursor *tcur)
305 {
306     int groupIndex = -1;
307     try
308     {
309         //was it in the viewport?

```

```

310     QPoint cursor = QPoint(tcur->getPosition().getScreenX(Tuio2Mudbox::m_aStatScreenResX), tcur->
311         getPosition().getScreenY(Tuio2Mudbox::m_aStatScreenResY));
312     QWidget *receiver = Kernel()->MainWindow()->childAt(cursor);
313     if (receiver->height() != Kernel()->ViewPort()->Height() || receiver->width() != Kernel()->
314         ViewPort()->Width())
315     {
316         //release mouse button to end previous stroke
317         if (triggerMouseReleaseButton(cursor.x(), cursor.y(), Qt::LeftButton, Qt::LeftButton) )
318         {
319             Kernel()->Log("%i Mouse released at X:%i Y:%i in removeTuioCursor() due to mouse release
320                 outside of viewport.\n", Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().
321                 getSeconds(), cursor.x(), cursor.y());
322         }
323         else
324         {
325             Kernel()->Log("%i Error releasing mouse at X:%i Y:%i in removeTuioCursor() due to mouse
326                 release outside of viewport.\n", Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().
327                 getSystemTime().getSeconds(), cursor.x(), cursor.y());
328         }
329     }
330     else {
331         groupIndex = Tuio2Mudbox::IdentifyCursorGroup(tcur);
332     }
333 }
334 catch (...) {
335     Kernel()->Log("warning: System exception in removeTuioCursor() at mouse release outside of
336         viewport.\n");
337 }
338 int numberOfCursors = -1;
339 if (groupIndex == 0 || groupIndex == 1) {
340     numberOfCursors = Tuio2Mudbox::NumberOfCursorsInGroup (groupIndex);
341 }
342 if(groupIndex == 0) { //strong hand
343     switch(numberOfCursors-1) { // -1 because the cursor is not deleted, yet. Here are actions listet
344         for the number of fingers remaining.
345         case 0 : MudboxTuioListener::selectTool("Not Mapped!");
346             break;
347         case 1 : MudboxTuioListener::selectTool("Knife");
348             break;
349         case 2 : MudboxTuioListener::selectTool("Sculpt");
350             break;
351         case 3 : {
352             //get angle withing group to detect aligned cursors.
353             TUIO::TuioPoint tpoint1 = Tuio2Mudbox::CursorGroups[groupIndex][1]->getPosition();
354             TUIO::TuioPoint tpoint2 = Tuio2Mudbox::CursorGroups[groupIndex][2]->getPosition();
355             float alpha1 = Tuio2Mudbox::CursorGroups[groupIndex][0]->getAngleDegrees(&tpoint1);
356             float alpha2 = Tuio2Mudbox::CursorGroups[groupIndex][0]->getAngleDegrees(&tpoint2);
357             float angle = fabs(alpha1 - alpha2);
358             if((fabs(angle) > 170 && fabs(angle) < 190 ) || fabs(angle) < 7 || fabs(angle) > 350) {
359                 //select eraser;
360                 MudboxTuioListener::selectTool("Eraser");
361             }
362             else
363             {
364                 //select Flatten
365                 MudboxTuioListener::selectTool("Flatten");
366             }
367             break;
368         }
369         case 4: MudboxTuioListener::selectTool("Grab");
370             break;
371         case 5: MudboxTuioListener::selectTool("Not Mapped!");
372             break;
373     }
374 }
375 else if(groupIndex == 1) { //weak hand
376     switch(numberOfCursors-1) { // -1 because the cursor is not deleted, yet. Here are actions listet
377         for the number of fingers remaining.
378         case 3 : if ((tcur->getTuioTime() - tcur->getStartTime()).getTotalMilliseconds() > 250){//if
379             only tapped toggle control key.
380             Tuio2Mudbox::inverted = !Tuio2Mudbox::inverted;
381             Tuio2Mudbox::inverted ? triggerKeyPressEvent(Qt::Key_Control) : triggerKeyReleaseEvent(Qt::
382                 Key_Control);
383             Kernel::Kernel().SetStatus(Kernel::stNormal, Tuio2Mudbox::inverted ? "brush inverted" : "brush
384                 de-inverted");
385         }
386         break;
387         case 4:
388             altPressed = false;
389             break;
390     }
391 }
392 }
393 Tuio2Mudbox::DeleteCursorFromGroups(tcur);
394 Tuio2Mudbox::redraw = true;
395 Kernel()->Redraw();

```

```

383 }
384
385 //when a cursor position is changed this is called
386 //here lies the logic to change and use tools
387 void MudboxTuioListener::updateTuioCursor(TUIO::TuioCursor *tcur)
388 {
389     int groupIndex = -1;
390     try {
391         //clicked into viewport?
392         QPoint cursor = QPoint(tcur->getPosition().getScreenX(Tuio2Mudbox::m_aStatScreenResX), tcur->
            getPosition().getScreenY(Tuio2Mudbox::m_aStatScreenResY));
393         QWidget *receiver = Kernel()->MainWindow()->childAt(cursor);
394         QPoint lastCursor = QPoint(tcur->getPath().back().getScreenX(Tuio2Mudbox::m_aStatScreenResX), tcur
            ->getPath().back().getScreenY(Tuio2Mudbox::m_aStatScreenResY));
395         QWidget *lastReceiver = Kernel()->MainWindow()->childAt(lastCursor);
396         if (receiver->height() != Kernel()->ViewPort()->Height() || receiver->width() != Kernel()->
            ViewPort()->Width())
397         {
398             //release mouse button to end previous stroke
399             if (!triggerMouseButtonRelease(cursor.x(), cursor.y(), Qt::LeftButton, Qt::LeftButton) )
400             {
401                 Kernel()->Log("%i Error moving mouse at X:%i Y:%i in updateTuioCursor() due to cursor
                    movement of viewport.\n", Tuio2Mudbox::CursorGroups[0][0]->getTuioTime().getSystemTime().
                    getSeconds(), cursor.x(), cursor.y());
402             }
403         }
404         else {
405             groupIndex = Tuio2Mudbox::IdentifyCursorGroup(tcur);
406         }
407     }
408     catch (...) {
409         Kernel()->Log("warning: System exception in updateTuioCursor() at movement outside of viewport due
            to widget change.\n");
410     }
411     //the following actions depend on the amount of cursors in the moving hand
412     int numberOfCursors = -1;
413     if (groupIndex == 0 || groupIndex == 1) {
414         numberOfCursors = Tuio2Mudbox::NumberOfCursorsInGroup (groupIndex);
415     }
416     try {
417         if(groupIndex == 0) { //strong hand
418             switch(numberOfCursors) {
419                 case 1 : MudboxTuioListener::selectTool(" Knife");
420                     break;
421                 case 2 : { //detect pinches and stretches with two fingers for the pinch and
422                         //smooth tools
423                     float xSp = tcur->getXSpeed();
424                     float ySp = tcur->getYSpeed();
425                     TUIO::TuioPoint delta = TUIO::TuioPoint(tcur->getX()+xSp, tcur->getY()+ySp);
426                     mudbox::Vector center = Tuio2Mudbox::getCenterOfGroup(groupIndex);
427                     TUIO::TuioPoint centerPoint = TUIO::TuioPoint(center.x, center.y);
428                     float alpha1 = tcur->getAngleDegrees(&delta);
429                     float alpha2 = tcur->getAngleDegrees(&centerPoint);
430                     float diff = fabs(alpha1 - alpha2);
431                     //pinch
432                     if ( fabs(diff) < 45 && tcur->getMotionSpeed() > Tuio2Mudbox::m_aStatToolSens && !
                        Tuio2Mudbox::isMovementUnified(tcur) ) {
433                         MudboxTuioListener::selectTool(" Pinch");
434                     }
435                     //stretch
436                     } else if ( fabs (diff) > 135 && fabs(diff) < 225 && tcur->getMotionSpeed() > Tuio2Mudbox::
                        m_aStatToolSens && !Tuio2Mudbox::isMovementUnified(tcur) ) {
437                         MudboxTuioListener::selectTool(" Smooth");
438                     }
439                     break;
440                 case 3 : { //depending on relative position and movement of the cursors a group
441                         //of three cursors means aligned=eraser, not aligned&no relative
442                         //movement=flatten and spread=foamy. Three finger pinch is not used.
443                     TUIO::TuioPoint tpoint1 = Tuio2Mudbox::CursorGroups[groupIndex][1]->getPosition();
444                     TUIO::TuioPoint tpoint2 = Tuio2Mudbox::CursorGroups[groupIndex][2]->getPosition();
445                     float alpha1 = Tuio2Mudbox::CursorGroups[groupIndex][0]->getAngleDegrees(&tpoint1);
446                     float alpha2 = Tuio2Mudbox::CursorGroups[groupIndex][0]->getAngleDegrees(&tpoint2);
447                     float angle = fabs(alpha1 - alpha2);
448                     //cursors aligned?
449                     if((fabs(angle) > 160 && fabs(angle) < 200 ) || fabs(angle) < 20 || fabs(angle) > 340) {
450                         MudboxTuioListener::selectTool(" Eraser");
451                     } else if (Tuio2Mudbox::selectedTool == " Eraser" ) {
452                         MudboxTuioListener::selectTool(" Flatten");
453                     } else {
454                         float xSp = tcur->getXSpeed();
455                         float ySp = tcur->getYSpeed();
456                         TUIO::TuioPoint delta = TUIO::TuioPoint(tcur->getX()+xSp, tcur->getY()+ySp);
457                         mudbox::Vector center = Tuio2Mudbox::getCenterOfGroup(groupIndex);
458                         TUIO::TuioPoint centerPoint = TUIO::TuioPoint(center.x, center.y);
459                         float alpha1 = tcur->getAngleDegrees(&delta);

```

```

460     float alpha2 = tcur->getAngleDegrees(&centerPoint);
461     float diff = fabs(alpha1 - alpha2);
462     //pinch
463     mudbox::Vector avgSpeed = Tuio2Mudbox::averageGroupSpeeds(groupIndex);
464     /* if ( fabs(diff) < 45 && tcur->getMotionSpeed() > Tuio2Mudbox::m_aStatToolSens && !
        Tuio2Mudbox::isMovementUnified(tcur) ) {
465         //mapping currently unused
466     } else */
467     //spread
468     if ( fabs (diff) > 135 && fabs(diff) < 225 && tcur->getMotionSpeed() > Tuio2Mudbox::
        m_aStatToolSens && !Tuio2Mudbox::isMovementUnified(tcur) ) {
469         MudboxTuioListener::selectTool("Foamy");
470     }
471 }
472 break;
473 }
474 case 4 : MudboxTuioListener::selectTool("Grab");
475 break;
476 case 5 : MudboxTuioListener::selectTool("Not Mapped!");
477 break;
478 }
479 //move cursor
480 //get data for mouse cursor
481 mudbox::Vector groupCenter = Tuio2Mudbox::getCenterOfGroup(groupIndex);
482 TUIO::TuioPoint loc = TUIO::TuioPoint(groupCenter.x, groupCenter.y);
483 QPoint glob(loc.getScreenX(Tuio2Mudbox::m_aStatScreenResX), loc.getScreenY(Tuio2Mudbox::
    m_aStatScreenResY));
484 int mouseButton1 = Qt::LeftButton;
485 int mouseButton2 = Qt::LeftButton;
486 if (numberOfCursors == 2 && altPressed)
487 {
488     mouseButton1 = Qt::MidButton;
489     mouseButton2 = Qt::MidButton;
490 }
491 else if (numberOfCursors == 3 && altPressed)
492 {
493     mouseButton2 = Qt::MidButton;
494 }
495 } else if (groupIndex == 1) { //weak hand
496 mudbox::Vector slideOffset;
497 switch(numberOfCursors) {
498     case 1 : { //one cursor on the weak hand has no effect
499         break;
500     }
501     case 2 : { //change tool size by pinching/stretching
502         if (Kernel()->m_pSelectedBrush) {
503             float xSp = tcur->getXSpeed();
504             float ySp = tcur->getYSpeed();
505             TUIO::TuioPoint delta = TUIO::TuioPoint(tcur->getX()+xSp, tcur->getY()+ySp);
506             mudbox::Vector center = Tuio2Mudbox::getCenterOfGroup(groupIndex);
507             TUIO::TuioPoint centerPoint = TUIO::TuioPoint(center.x, center.y);
508             float alpha1 = tcur->getAngleDegrees(&delta);
509             float alpha2 = tcur->getAngleDegrees(&centerPoint);
510             float diff = fabs(alpha1 - alpha2);
511             //pinch
512             mudbox::Vector avgSpeed = Tuio2Mudbox::averageGroupSpeeds(groupIndex);
513             if ( fabs(diff) < 45 && tcur->getMotionSpeed() > Tuio2Mudbox::m_aStatParamSens) {
514                 float newSize = Kernel()->m_pSelectedBrush->BrushSize() - (tcur->getMotionSpeed()*7);
515                 if (newSize < 0) {
516                     newSize = 0;
517                 } else if (newSize > 100) {
518                     newSize = 100;
519                 }
520                 Kernel()->m_pSelectedBrush->SetBrushSize(newSize);
521                 Kernel()->SetStatus(Kernel::stNormal, "new size: " + QString().sprintf("%.3f", newSize));
522             }
523             //stretch
524             } else if (fabs (diff) > 135 && fabs(diff) < 225 && tcur->getMotionSpeed() > Tuio2Mudbox::
                m_aStatParamSens) {
525                 float newSize = Kernel()->m_pSelectedBrush->BrushSize() + (tcur->getMotionSpeed()*7);
526                 if (newSize < 0) {
527                     newSize = 0;
528                 } else if (newSize > 100) {
529                     newSize = 100;
530                 }
531                 Kernel()->m_pSelectedBrush->SetBrushSize(newSize);
532                 Kernel()->SetStatus(Kernel::stNormal, "new size: " + QString().sprintf("%.3f", newSize));
533             }
534         }
535         break;
536     }
537     case 3 : { //change tool strength by pinching/spreading with three fingers
538         if (Kernel()->m_pSelectedBrush) {
539             float a = tcur->getXSpeed();

```

```

539     float b = tcur->getYSpeed();
540     TUIO::TuioPoint delta = new TUIO::TuioPoint(tcur->getX()+tcur->getXSpeed(), tcur->getY()+
541         tcur->getYSpeed());
542
543     mudbox::Vector center = Tuio2Mudbox::getCenterOfGroup(groupIndex);
544     TUIO::TuioPoint centerPoint = TUIO::TuioPoint(center.x, center.y);
545     float alpha1 = tcur->getAngleDegrees(&delta);
546     float alpha2 = tcur->getAngleDegrees(&centerPoint);
547     float diff = fabs(alpha1 - alpha2);
548     //pinch
549     mudbox::Vector avrgSpeed = Tuio2Mudbox::averageGroupSpeeds(groupIndex);
550     if ( fabs(diff) < 45 && tcur->getMotionSpeed() > Tuio2Mudbox::m_aStatParamSens) {
551         float newStrength = Kernel()->m_pSelectedBrush->BrushStrength() - (tcur->getMotionSpeed
552             (*7));
553         if (newStrength > 100) {
554             newStrength = 100;
555         } else if (newStrength < 0) {
556             newStrength = 0;
557         }
558         Kernel()->m_pSelectedBrush->SetBrushStrength(newStrength);
559         Kernel()->SetStatus(Kernel::stNormal, "new strength: " + QString().sprintf("%.3f",
560             newStrength));
561     } //spread
562     } else if (fabs(diff) > 135 && fabs(diff) < 225 && tcur->getMotionSpeed() > Tuio2Mudbox::
563         m_aStatParamSens) {
564         float newStrength = Kernel()->m_pSelectedBrush->BrushStrength() + (tcur->getMotionSpeed
565             (*7));
566         if (newStrength > 100) {
567             newStrength = 100;
568         } else if (newStrength < 0) {
569             newStrength = 0;
570         }
571         Kernel()->m_pSelectedBrush->SetBrushStrength(newStrength);
572         Kernel()->SetStatus(Kernel::stNormal, "new strength: " + QString().sprintf("%.3f",
573             newStrength));
574     }
575     }
576     }
577     }
578     }
579     catch (...) {
580         Kernel()->Log("warning: System exception in updateTuioCursor() at gesture handling.\n");
581     }
582
583     Tuio2Mudbox::redraw = true;
584     Kernel()->Redraw();
585 }
586
587 //structurally needed, but has nothing to do
588 void MudboxTuioListener::addTuioObject(TUIO::TuioObject *tobj)
589 {
590 }
591
592 //structurally needed, but has nothing to do
593 void MudboxTuioListener::removeTuioObject(TUIO::TuioObject *tobj)
594 {
595 }
596
597 //structurally needed, but has nothing to do
598 void MudboxTuioListener::updateTuioObject(TUIO::TuioObject *tobj)
599 {
600 }
601
602 //structurally needed, but has nothing to do
603 void MudboxTuioListener::refresh(TUIO::TuioTime frameTime)
604 {
605 }

```

./c++/MudboxTuioListener.cpp