Documentation For
Cantor Set Analysis and Visualization with ADSODA
In 2, 3, and 4 dimensions

Fabian Junge

Friday December 21st, 2012

Contents

1 The RTICAs **3**
1.1 Location . . . 3
1.2 Running / Demoing . . . 3
1.2.1 adsoda_conversion . . . 3
1.2.2 adsoda_cantor . . . 3
1.3 Editing and Recompiling . . . 4
1.3.1 adsoda_cantor . . . 4
1.3.2 adsoda_conversion . . . 5

2 Hand Annotated RTICA Code **6**
2.1 adsoda_cantor . . . 6
2.1.1 adsoda_cantor.cpp . . . 6
2.1.2 ca_adsoda_cantorlevel.h . . . 17
2.1.3 ca_adsoda_cantorlevel.cpp . . . 19
2.2 adsoda_conversion . . . 29
2.2.1 cc_adsoda.cpp . . . 29
2.2.2 cc_adsoda_demo.cpp . . . 34

1 The RTICAs

The main RTICA, shown in the final presentation, for this project is "adsoda_cantor".
There is also a side RTICA "cc_adsoda" which simply demonstrates the successful porting of ADSODA.

1.1 Location

Both RTICAs are located in fjunge2/cubeready, adsoda_cantor being in the folder "adsoda_cantor", cc_adsoda residing in the folder "adsoda_conversion".
These two folders have the same folder structure as standard Aszgard/Syzygy applications. That is, there is a src/ folder with the source .cpp and .h codes and the makefile for the project is located in "build/makefiles" being titled "Makefile.my_app".

1.2 Running / Demoing

In the "cubeready/" folder there is a README file which explains how the RTICAs run, the information here is directly copied from it.

1.2.1 adsoda_conversion

This was getting ADSODA to work with C++03 and our Cube.
Making and running it will display one of ADSODA's demos, two 4D rotated hypercubes.
There is a white outline of sides on the hypercubes, this can be toggled off/on by pressing 's'.
There is no cube button to toggle the outline, as this was written before I figured how to use cube buttons.
The hypercubes are both rotated by rotating:
The x-w plane "0" radians counter-clockwise,
The y-w plane "2" radians counter-clockwise, and
The z-w plane "0" radians counter-clockwise.

1.2.2 adsoda_cantor

The program displays each of the 2, 3, and 4 dimensional representations of the Cantor Set.
The program starts out displaying the 2 dimensional Sierpinski Carpet at recursion level 1 in the center of the screen.
Movement is done with the Joystick of the wand, and the fractal displayed can be changed by the following keys / buttons.

Table 1: Keys

The 'r' key will	increase the recursion level by 1, to a maximum of 6 in the 2d case, and no maximums in the 3d or 4d cases.
The 'e' key will:	decrease the recursion level by 1, to a minimum of 0.
The 'd' key will:	increase the dimension by 1, to a maximum of 4.
The 's' key will:	decrease the dimension level by 1, to a minimum of 2.
The 'c' key will:	change the solid being used, but will not change the dimension (IE from Sierpinski Carpet to 2D dust).
The 'x' key will	toggle between showing just the outlines of the solids or filling them in.

Table 2: Cube Wand Buttons

3		: Increase the Dimension of the Solids
0		: Decrease the Dimension of the Solids
1		: Increase the Recursion Level
2		: Decrease the Recursion Level
4	(Trigger)	: Toggles whether or not the solids are filled or simply outlined
5	(Clicking in Joystick)	: Change the solid being displayed (in that dimension, see 'c' key above)

Note: Changing the solid or changing the dimension will reset the recursion level back to level 1. This occurs on both the keypresses and the Wand.

IMPORTANT: If you press "Increase Recursion" and the program seems to freeze, it is highly likely that it is just taking a long time to compute the next level of recursion.
You can double check this by trying to move, if you cannot move, the program is calculating.
For higher recursion levels (especially in when in three or four dimensions) calculating the display can take quite a while.
DO NOT PRESS THE BUTTON AGAIN THINKING IT DIDN'T REGISTER.
If you do, it will start to calculate the next level after it is finished with the current one and take even longer. "

1.3 Editing and Recompiling

As a note, for both of the RTICAs, if you change any file other than "cc_adsoda.cpp" or "adsoda_cantor.cpp", you will have to execute a "make clean" before you execute "make" for the changes to take effect.

1.3.1 adsoda_cantor

To simply compile the RTICA, navigate to the main folder ("cubeready/adsoda_cantor") and execute the "make" command.
If you wish to edit the code, there are three main files of interest (these are, coincidentally, the three non-ADSODA files).

- "src/adsoda_cantor.cpp": This file contains the main Syzygy application with the values inherent to one (key bindings, button bindings, navigation speeds, near clipping plane distance, far clipping plane distance, etc.).
  This file also contains some general logic of the program, such as where the solids are centered, the limits of recursion, the order in which "change solid" cycles, minor things like that.

- "src/ca_adsoda_cantorlevel.cpp": This file contains the main code for each of the solids displayed, such as the algorithm for creating the next level of recursion, the algorithm for rotating the 4d cubes, and the algorithm for drawing and displaying the whole thing.

- "src/ca_adsoda_cantorlevel.h": This file is mainly interesting in that it is the header file for the above .cpp. If you wish to add any functions to the .cpp above or any constants to the namespace the .cpp above resides in, you should add them here as well.

All three of these files will be included and hand annotated later on.

1.3.2 adsoda_conversion

ADSODA, in it's original form, had the ability to run different demos depending on command line input. In the ported version, the settings that could be set by the command line are set in the constructor of the SkeletonFramework. This function currently appears as so:

```
164 SkeletonFramework::SkeletonFramework() :
       arMasterSlaveFramework()//, _squareHighlightedTransfer(0)
165 {
166     /* Nothing Here */
167     initState(s_state);
168     // parseArgs(s_state, argc, argv);
169     s_state.dimension = 4;
170     s_state.draw3D = true;
171     s_state.rotate4D = true;
172
173     s_state.theta = 1;
174     s_state.rho = 0;
175     s_state.phi = 0;
176     s_state.removeHidden4D = true;
177     s_state.outlinePolys = true;
178 }
```

For a list of available settings, take a look at "adsoda_conversion/src/cc_adsoda_state":

```
 6 typedef struct {
 7
 8   Space *demoSpace;
 9   Space *draw1DSpace;
10   Space *draw2DSpace;
11   Space *draw3DSpace;
12
13   bool outlinePolys;
14   bool fillPolys;
15   int dimension;
16   bool draw1D;
17   bool draw2D;
18   bool draw3D;
19   bool removeHidden2D;
20   bool removeHidden3D;
21   bool removeHidden4D;
22   bool rotate2D;
23   bool rotate3D;
24   bool rotate4D;
25   bool demoInitialized;
26   bool drawcubeFlag;
27   double theta;
28   double rho;
29   double phi;
30
31 } State;
```

To change how the hypercubes are rotated, s_state.rho rotates the x-w plane, s_state.theta the y-w plane and s_state.phi the z-w plane.
The cc_adsoda.cpp file also appears annotated below.

2 Hand Annotated RTICA Code

Note that, to save space, I have cut out some swaths of commented out code throughout several of these files. This will be represented by a ... and a skip in the line numbers on the left.

2.1 adsoda_cantor

2.1.1 adsoda_cantor.cpp

```
1  //*************************************************
2  // Syzygy is licensed under the BSD license v2
3  // see the file SZG_CREDITS for details
4  //*************************************************
5
6  // precompiled header include MUST appear as the first
       non-comment line
7  #include "arPrecompiled.h"
8  // MUST come before other szg includes. See
       arCallingConventions.h for details.
9  #define SZG_DO_NOT_EXPORT
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <vector>
13 #include <iostream>
14 #include <math.h>
15 #include "arMasterSlaveFramework.h"
16 #include "arInteractableThing.h"
17 #include "arInteractionUtilities.h"
18 #include "arGlut.h"
19
20 // ADSODA Includes
21 #include "ca_adsoda_cantorlevel.h"
22
23 using cLevel::cantorLevel;
24 using namespace std;

   ...

40 const float FEET_TO_LOCAL_UNITS = 1.;
41
42 // Near & far clipping planes.
43 const float nearClipDistance = .1*FEET_TO_LOCAL_UNITS;
44 const float farClipDistance = 100.*FEET_TO_LOCAL_UNITS;
45
```

```
46 class cantorInterface
47 {
48 public:
49     cantorInterface(int sideLength);
50     cantorInterface(int dim, int recLevel, const arMatrix4 &
          viewpoint, int sideLength);
51
52     void setMatrix(const arMatrix4 & view) {viewpoint = view;};
53     void draw();
54
55     void increaseRecurse();
56     void decreaseRecurse();
57     void increaseDimension();
58     void decreaseDimension();
59     void changeSolid();
60     void toggleOutline();
61
62 private:
63     int curDimension;
64     int curRecursionLevel;
65     int curSolid;
66     bool outline;
67     arMatrix4 viewpoint;
68     cantorLevel<2, cLevel::I> carpet;
69     cantorLevel<2, cLevel::U> dust2;
70     cantorLevel<3, cLevel::UU> dust3;
71     cantorLevel<3, cLevel::II> enclosed3;
72     cantorLevel<3, cLevel::IU> mengersponge;
73     cantorLevel<4, cLevel::UUU> dust4;
74     cantorLevel<4, cLevel::III> enclosed4;
75     cantorLevel<4, cLevel::UUI> menger4pos1;
76     cantorLevel<4, cLevel::IIU> menger4pos2;
77
78     void choseBasicSolid();
79 };
80
81 cantorInterface::cantorInterface(int sideLength) :
82   carpet(sideLength), dust2(sideLength), dust3(sideLength),
83   enclosed3(sideLength), mengersponge(sideLength),
       dust4(sideLength),
84   enclosed4(sideLength), menger4pos1(sideLength),
       menger4pos2(sideLength)
85 {
86     curDimension = 2;
87     curRecursionLevel = 1;
88     curSolid = cLevel::I;
89     outline = true;
90 }
```

```
91
92  cantorInterface::cantorInterface(int dim, int recLevel, const
        arMatrix4 & view, int sideLength) :
93    carpet(sideLength), dust2(sideLength), dust3(sideLength),
94    enclosed3(sideLength), mengersponge(sideLength),
          dust4(sideLength),
95    enclosed4(sideLength), menger4pos1(sideLength),
          menger4pos2(sideLength)
96  {
97      curDimension = dim;
98      curRecursionLevel = recLevel;
99      viewpoint = view;
100     curSolid = cLevel::I;
101     outline = true;
102 }
103
104 void cantorInterface::increaseRecurse()
105 {
106     if (curDimension == 2)
107     {
108         if (curRecursionLevel < 6)
109         {
110             curRecursionLevel++;
111             if (curSolid == cLevel::I)
112                 carpet.generateRecLevel(curRecursionLevel);
113             else if (curSolid == cLevel::U)
114                 dust2.generateRecLevel(curRecursionLevel);
115         }
116     }
117     else if (curDimension == 3)
118     {
119         // No limit set yet.
120         curRecursionLevel++;
121         if (curSolid == cLevel::UU)
122             dust3.generateRecLevel(curRecursionLevel);
123         else if (curSolid == cLevel::II && curRecursionLevel <
              4)
124             enclosed3.generateRecLevel(curRecursionLevel);
125         else if (curSolid == cLevel::IU)
126             mengersponge.generateRecLevel(curRecursionLevel);
127     }
128     else if (curDimension == 4)
129     {
130         // No limit set yet.
131         curRecursionLevel++;
132         if (curSolid == cLevel::UUU)
133             dust4.generateRecLevel(curRecursionLevel);
134         else if (curSolid == cLevel::III)
```

```
135             enclosed4.generateRecLevel(curRecursionLevel);
136         else if (curSolid == cLevel::UUI)
137             menger4pos1.generateRecLevel(curRecursionLevel);
138         else if (curSolid == cLevel::IIU)
139             menger4pos2.generateRecLevel(curRecursionLevel);
140     }
141 }
142
143 void cantorInterface::decreaseRecurse()
144 {
145     if (curRecursionLevel > 0)
146         curRecursionLevel--;
147 }
148
149 void cantorInterface::increaseDimension()
150 {
151     if (curDimension < 4)
152     {
153         curDimension++;
154         curRecursionLevel = 1;
155         choseBasicSolid();
156         // if (curDimension == 2)
157         //     setMatrix(ar_translationMatrix(-32, -28, -15));
158         // else if (curDimension == 3)
159         //     setMatrix(ar_translationMatrix(-32, -28, -58));
160         // else if (curDimension == 4)
161         //     setMatrix(ar_translationMatrix(33, -28, -58));
162     }
163 }
164
165 void cantorInterface::decreaseDimension()
166 {
167     if (curDimension > 2)
168     {
169         curDimension--;
170         curRecursionLevel = 1;
171         choseBasicSolid();
172         // if (curDimension == 2)
173         //     setMatrix(ar_translationMatrix(-32, -28, -15));
174         // else if (curDimension == 3)
175         //     setMatrix(ar_translationMatrix(-32, -28, -58));
176         // else if (curDimension == 4)
177         //     setMatrix(ar_translationMatrix(35, -28, -58));
178     }
179 }
180
181 void cantorInterface::changeSolid()
182 {
```

```
183     if (curSolid == cLevel::U)
184         curSolid = cLevel::I;
185     else if (curSolid == cLevel::I)
186         curSolid = cLevel::U;
187     else if (curSolid == cLevel::UU)
188         curSolid = cLevel::II;
189     else if (curSolid == cLevel::II)
190         curSolid = cLevel::IU;
191     else if (curSolid == cLevel::IU)
192         curSolid = cLevel::UU;
193     else if (curSolid == cLevel::UUU)
194         curSolid = cLevel::III;
195     else if (curSolid == cLevel::III)
196         curSolid = cLevel::UUI;
197     else if (curSolid == cLevel::UUI)
198         curSolid = cLevel::IIU;
199     else if (curSolid == cLevel::IIU)
200         curSolid = cLevel::UUU;
201     else
202         curSolid = cLevel::U;
203     curRecursionLevel = 1;
204 }
205
206 void cantorInterface::toggleOutline()
207 {
208     outline = !outline;
209 }
210
211 void cantorInterface::choseBasicSolid()
212 {
213     if (curDimension == 2)
214     {
215         curSolid = cLevel::I;
216     }
217     else if (curDimension == 3)
218     {
219         curSolid = cLevel::UU;
220     }
221     else
222     {
223         curSolid = cLevel::UUU;
224     }
225 }
226
227 void cantorInterface::draw()
228 {
229     glMultMatrixf(viewpoint.v);
230
```

```
231     if (curDimension == 2)
232     {
233         if (curSolid == cLevel::U)
234         {
235             dust2.draw(curRecursionLevel, outline);
236         }
237         else
238         {
239             carpet.draw(curRecursionLevel, outline);
240         }
241     }
242     else if (curDimension == 3)
243     {
244         if (curSolid == cLevel::UU)
245         {
246             dust3.draw(curRecursionLevel, outline);
247         }
248         else if (curSolid == cLevel::II)
249         {
250             enclosed3.draw(curRecursionLevel, outline);
251         }
252         else if (curSolid == cLevel::IU)
253         {
254             mengersponge.draw(curRecursionLevel, outline);
255         }
256     }
257     else if (curDimension == 4)
258     {
259         if (curSolid == cLevel::UUU)
260             dust4.draw(curRecursionLevel, outline);
261         else if (curSolid == cLevel::III)
262             enclosed4.draw(curRecursionLevel, outline);
263         else if (curSolid == cLevel::UUI)
264             menger4pos1.draw(curRecursionLevel, outline);
265         else if (curSolid == cLevel::IIU)
266             menger4pos2.draw(curRecursionLevel, outline);
267     }
268 }
269
270 class CantorFramework: public arMasterSlaveFramework
271 {
272 public:
273     CantorFramework();
274     bool onStart( arSZGClient& SZGClient );
275     void onWindowStartGL( arGUIWindowInfo* );
276     void onPreExchange( void );
277     void onPostExchange( void );
278 //    void onWindowInit( void );
```

```
279     void onDraw( arGraphicsWindow& win, arViewport& vp );
280 //    void onDisconnectDraw( void );
281 //    void onPlay( void );
282     void onWindowEvent( arGUIWindowInfo* );
283 //    void onCleanup( void );
284 //    void onUserMessage( const string& messageBody );
285 //    void onOverlay( void );
286 //    void onKey( unsigned char key, int x, int y );
287     void onKey( arGUIKeyInfo* );
288 //    void onMouse( arGUIMouseInfo* );
289 private:
    ...
300     cantorInterface cantor;
301
302     // State s_state;
303 };
304
305
306 CantorFramework::CantorFramework() : arMasterSlaveFramework(),
      cantor(2187)//, _squareHighlightedTransfer(0)
307 {
    ...
314 }
315
316
317 // onStart callback method (called in
      arMasterSlaveFramework::start()
318 //
319 // Note: DO NOT do OpenGL initialization here; this is now
      called
320 // __before__ window creation. Do it in the onWindowStartGL()
321 //
322 bool CantorFramework::onStart( arSZGClient& /*cli*/ )
323 {
    ...
329     // Setup navigation, so we can drive around with the
        joystick
330     //
331     // Tilting the joystick by more than 20% along axis 1 (the
        vertical on ours) will cause
332     // translation along Z (forwards/backwards). This is
        actually the default behavior, so this
333     // line isn't necessary.
334     setNavTransCondition( 'z', AR_EVENT_AXIS, 1, 0.2 );
335
336     // Tilting joystick left or right will rotate left/right
        around vertical axis (default is left/right
```

```
337     // translation)
338     setNavRotCondition( 'y', AR_EVENT_AXIS, 0, 0.2 );
339
340     // Set translation & rotation speeds to 5 ft/sec & 30
           deg/sec (defaults)
341     setNavTransSpeed( 5. );
342     setNavRotSpeed( 30. );
343
344     // set square's initial position
345     // _square.setMatrix( ar_translationMatrix(0,5,-6) );
346
347     // Set cantor's initial position
348     // cantor.setMatrix(ar_translationMatrix(-32, -28, -15));
349     cantor.setMatrix(ar_translationMatrix(-10, -6, -12));
350
351
352     return true;
353 }
354
355 // Method to initialize each window (because now a Syzygy app
       can
356 // have more than one).
357 void CantorFramework::onWindowStartGL( arGUIWindowInfo* )
358 {
359     // OpenGL initialization
360     glClearColor(0,0,0,0);
361 }
362
363
364 // Method called before data is transferred from master to
       slaves. Only called
365 // on the master. This is where anything having to do with
366 // processing user input or random variables should happen.
367 void CantorFramework::onPreExchange()
368 {
369     // Do stuff on master before data is transmitted to slaves.
370
371     // handle joystick-based navigation (drive around). The
           resulting
372     // navigation matrix is automagically transferred to the
           slaves.
373     navUpdate();
374 }
    ...
388 // Method called after transfer of data from master to slaves.
       Mostly used to
389 // synchronize slaves with master based on transferred data.
390 void CantorFramework::onPostExchange()
```

```
391 {
392     // Do stuff after slaves got data and are again in sync
        with the master.
393    // if (!getMaster())
394    //
395     // Buttons!
396     const int butn[8] = {getButton(0), getButton(1),
        getButton(2), getButton(3), getButton(4), getButton(5),
397                          getButton(6), getButton(7)};
398     static int obutn[8] = {0};
399
400     if (obutn[1] == 0 && butn[1] == 1)
401         cantor.increaseRecurse();
402     if (obutn[2] == 0 && butn[2] == 1)
403         cantor.decreaseRecurse();
404     if (obutn[3] == 0 && butn[3] == 1)
405         cantor.increaseDimension();
406     if (obutn[0] == 0 && butn[0] == 1)
407         cantor.decreaseDimension();
408     if (obutn[5] == 0 && butn[5] == 1)
409         cantor.changeSolid();
410     if (obutn[4] == 0 && butn[4] == 1)
411         cantor.toggleOutline();
412     memcpy(obutn, butn, 8*sizeof(int));
413 }
414
415 void CantorFramework::onDraw( arGraphicsWindow& /*win*/,
    arViewport& /*vp*/ )
416 {
417     // Load the navigation matrix.
418     loadNavMatrix();

    ...

423     glPushMatrix();

    ...

429     cantor.draw();
430
431     glPopMatrix();
432
433 }
434
435 // Catch key events.
436 void CantorFramework::onKey( arGUIKeyInfo* keyInfo )
437 {
438     cout << "Key␣ascii␣value␣=␣" << keyInfo->getKey() << endl;
439     cout << "Key␣ctrl␣␣␣␣value␣=␣" << keyInfo->getCtrl() <<
        endl;
```

```
440     cout << "Key␣alt␣␣␣␣␣␣value␣=␣" << keyInfo->getAlt() <<
            endl;
441     string stateString;
442     arGUIState state = keyInfo->getState();
443     if (state == AR_KEY_DOWN)
444     {
445         stateString = "DOWN";
446         arGUIKey key = keyInfo->getKey();
447         if (key == 'r') // r
448         {
449             cantor.increaseRecurse();
450         }
451         else if (key == 'e') // e
452         {
453             cantor.decreaseRecurse();
454         }
455         else if (key == 'd') // d
456         {
457             cantor.increaseDimension();
458         }
459         else if (key == 's') // s
460         {
461             cantor.decreaseDimension();
462         }
463         else if (key == 'c') // c
464         {
465             cantor.changeSolid();
466         }
467         else if (key == 'x')
468         {
469             cantor.toggleOutline();
470         }
471     }
472     else if (state == AR_KEY_UP)
473     {
474         stateString = "UP";
475     }
476     else if (state == AR_KEY_REPEAT)
477     {
478         stateString = "REPEAT";
479     }
480     else
481     {
482         stateString = "UNKNOWN";
483     }
484     cout << "Key␣state␣=␣" << stateString << endl;
485 }
486
```

```
487 // This is how we have to catch reshape events now, still
488 // dealing with the fallout from the GLUT->arGUI conversion.
489 // Note that the behavior implemented below is the default.
490 void CantorFramework::onWindowEvent( arGUIWindowInfo* winInfo )
491 {
492     // The values are defined in src/graphics/arGUIDefines.h.
493     // arGUIWindowInfo is in arGUIInfo.h
494     // The window manager is in arGUIWindowManager.h
495     if (winInfo->getState() == AR_WINDOW_RESIZE)
496     {
497         const int windowID = winInfo->getWindowID();
498 
499         #ifdef UNUSED
500             const int x = winInfo->getPosX();
501             const int y = winInfo->getPosY();
502         #endif
503 
504         const int width = winInfo->getSizeX();
505         const int height = winInfo->getSizeY();
506         getWindowManager()->setWindowViewport( windowID, 0, 0,
               width, height );
507     }
508 }
509 
510 int main(int argc, char** argv)
511 {
512     CantorFramework framework;
513     // Tell the framework what units we're using.
514     framework.setUnitConversion(FEET_TO_LOCAL_UNITS);
515     framework.setClipPlanes(nearClipDistance, farClipDistance);
516 
517     if (!framework.init(argc, argv))
518     {
519         return 1;
520     }
521 
522     // Never returns unless something goes wrong
523     return framework.start() ? 0 : 1;
524 }
```

2.1.2 ca_adsoda_cantorlevel.h

```
1  #ifndef HCANTOR_LEVEL
2  #define HCANTOR_LEVEL
3
4  #include "arPrecompiled.h"
5  #include <vector>
6  #include <iostream>
7  #include <cmath>
8  #include "ca_adsoda_solid.h"
9  #include "ca_adsoda_light.h"
10 #include "ca_adsoda_color.h"
11 #include "ca_adsoda_amatrix.h"
12 #include "arGlut.h"
13 #include "ca_adsoda_util.h"
14
15 #ifdef _OPENMP
16 #include <omp.h>
17 #endif
18
19 using std::vector;
20 using std::cerr;
21
22 namespace cLevel
23 {
24     template<int dim, int mode>
25     class cantorLevel
26     {
27     public:
28         /**
29          * Creates a cantor level at recursion level 0.
30          * This is the only public constructor, for higher
               recursion levels,
31          * an internal constructor is used.
32          */
33         cantorLevel(int side);
34         ~cantorLevel();
35
36         void draw(int recLevel, bool outline);
37         void generateRecLevel(int recL);
38
39     private:
40         int rec;
41         bool displayGen, childGen;
42         int sideLength;
43         GLuint displayList;
44         GLuint displayListOutline;
45         GLuint innerList;
```

```
46          cantorLevel<dim, mode> * child;
47          vector<Solid *> solids;
48
49          void generateDisplay();
50
51          void generateChild();
52
53          void freeSolids();
54
55          cantorLevel(int recLevel, vector<Solid *> &
               parentSolids, int sideL);
56
57          void splitSolid(Solid & base, int sideL);
58
59          bool skip(int xPos, int yPos, int zPos, int wPos);
60
61      };
62
63      const int I = 0;
64      const int U = 1;
65      const int II = 2;
66      const int UU = 3;
67      const int IU = 4;
68      const int UUU = 5;
69      const int III = 6;
70      const int UUI = 7;
71      const int IIU = 8;
72      const double PI = 3.14159265359;
73  }
74  #include "ca_adsoda_cantorlevel.cpp"
75  #endif
```

2.1.3 ca_adsoda_cantorlevel.cpp

```
1  #ifndef CPPCANTOR_LEVEL
2  #define CPPCANTOR_LEVEL
3
4  #include "ca_adsoda_cantorlevel.h"
5
6  using namespace cLevel;
7
8  /**
9   * Public Constructor
10  */
11 template<int dim, int mode>
12 cantorLevel<dim, mode>::cantorLevel(int side)
13 {
14     displayGen = false;
15     childGen = false;
16     child = NULL;
17     rec = 0;
18     sideLength = side;
19
20     // Generate basic Solid
21     solids.reserve(1);
22     // double min = -(side/2);
23     // double max = side/2;
24     double min = 0;
25     double max = side;
26     if (dim == 2)
27     {
28         solids.push_back(Util::NewSquare(min, max, min, max));
29         solids[0]->SetColor(0, 1, 1);
30     }
31     else if (dim == 3)
32     {
33         solids.push_back(Util::New3Cube(min, max, min, max,
              min, max));
34         solids[0]->SetColor(0, 1, 1);
35     }
36     else if (dim == 4)
37     {
38         solids.push_back(Util::New4Cube(min, max, min, max,
              min, max, min, max));
39         solids[0]->SetColor(0, 1, 1);
40     }
41 }
42
43 /**
44  * Internal Constructor
```

```
45  */
46 template<int dim, int mode>
47 cantorLevel<dim, mode>::cantorLevel(int recLevel,
      std::vector<Solid *> & parentSolids, int sideL)
48 {
49     rec = recLevel;
50     displayGen = false;
51     childGen = false;
52     child = NULL;
53     sideLength = sideL;
54
55     // Generate all of the solids at this level of recursion
56     if (dim == 2)
57     {
58         // if (mode == U)
59         //     solids.reserve((int)std::pow((double)4, rec));
60         // else if (mode == I)
61         //     solids.reserve((int)std::pow((double)8, rec));
62     }
63     else if (dim == 3)
64     {
65         // if (mode == U)
66         //     solids.reserve((int)std::pow((double)8, rec));
67         // else if (mode == II)
68         //     solids.reserve((int)std::pow((double)26, rec));
69         // else if (mode == IU)
70         //     solids.reserve((int)std::pow((double)20, rec));
71     }
72     else
73     {
74         // if (mode == UUU)
75             // solids.reserve((int)std::pow((double)8, rec));
76         // if (mode == IIU)
77             // solids.reserve((int)std::pow((double)72, rec));
78     }
79
80     std::vector<Solid *>::iterator it = parentSolids.begin();
81     std::vector<Solid *>::iterator end = parentSolids.end();
82     std::cerr << "Generating␣Children:" << std::endl;
83     #ifdef _OPENMP
84     double startime = omp_get_wtime();
85     #pragma omp parallel for
86     #endif
87     for (int i = 0; it != end; it++, i++)
88     {
89         splitSolid(**it, sideLength);
90     }
91     #ifdef _OPENMP
```

```
92      double timetaken = omp_get_wtime() - startime;
93      std::cerr << "Time Taken:" << timetaken << std::endl;
94      #endif
95  }
96
97  /**
98   * Destructor
99   */
100 template<int dim, int mode>
101 cantorLevel<dim, mode>::~cantorLevel()
102 {
103     freeSolids();
104     glDeleteLists(displayList, 1);
105     glDeleteLists(displayListOutline, 1);
106     glDeleteLists(innerList, 1);
107     delete child; // Which calls child's destructor, which
          call child's child's destructor, etc.
108 }
109
110 /**
111  * Destructor (and elsewhere) helper
112  */
113 template<int dim, int mode>
114 void cantorLevel<dim, mode>::freeSolids()
115 {
116     vector<Solid *>::iterator it = solids.begin();
117     vector<Solid *>::iterator end = solids.end();
118     for(; it != end; ++it)
119     {
120         delete *it;
121         *it = NULL;
122     }
123     solids.clear();
124 }
125
126 template<int dim, int mode>
127 void cantorLevel<dim, mode>::splitSolid(Solid & base, int
      sideL)
128 {
129     std::vector<double> minimums = base.getMinimums();
130
131     if (dim == 2)
132     {
133         for (int x = 0; x < 3; x++)
134         {
135             double xmin = minimums[0] + (x * sideL);
136             double xmax = xmin + sideL;
137             for (int y = 0; y < 3; y++)
```

```
138             {
139                 double ymin = minimums[1] + (y * sideL);
140                 double ymax = ymin + sideL;
141 
142                 if (skip(x, y, 0, 0))
143                     continue;
144                 Solid * sod = Util::NewSquare(xmin, xmax,
                       ymin, ymax);
145                 Color scolor;
146                 base.GetColor(scolor);
147                 sod->SetColor(scolor);
148                 solids.push_back(sod);
149             }
150         }
151     }
152     else if (dim == 3)
153     {
154         for (int x = 0; x < 3; x++)
155         {
156             double xmin = minimums[0] + (x * sideL);
157             double xmax = xmin + sideL;
158             for (int y = 0; y < 3; y++)
159             {
160                 double ymin = minimums[1] + (y * sideL);
161                 double ymax = ymin + sideL;
162                 for (int z = 0; z < 3; z++)
163                 {
164                     double zmin = minimums[2] + (z * sideL);
165                     double zmax = zmin + sideL;
166                     if (skip(x, y, z, 0))
167                         continue;
168                     Solid *sod = Util::New3Cube(xmin, xmax,
                           ymin, ymax, zmin, zmax);
169                     Color scolor;
170                     base.GetColor(scolor);
171                     sod->SetColor(scolor);
172                     solids.push_back(sod);
173                 }
174             }
175         }
176     }
177     else if (dim == 4)
178     {
179         for (int x = 0; x < 3; x++)
180         {
181             double xmin = minimums[0] + (x * sideL);
182             double xmax = xmin + sideL;
183             for (int y = 0; y < 3; y++)
```

```
184             {
185                 double ymin = minimums[1] + (y * sideL);
186                 double ymax = ymin + sideL;
187                 for (int z = 0; z < 3; z++)
188                 {
189                     double zmin = minimums[2] + (z * sideL);
190                     double zmax = zmin + sideL;
191                     for (int w = 0; w < 3; w++)
192                     {
193                         double wmin = minimums[3] + (w *
                               sideL);
194                         double wmax = wmin + sideL;
195                         if (skip(x, y, z, w))
196                             continue;
197                         Solid *sod = Util::New4Cube(xmin,
                               xmax, ymin, ymax, zmin, zmax, wmin,
                               wmax);
198                         Color scolor;
199                         base.GetColor(scolor);
200                         sod->SetColor(scolor);
201                         solids.push_back(sod);
202                     }
203                 }
204             }
205         }
206     }
207 }
208
209 template<int dim, int mode>
210 bool cantorLevel<dim, mode>::skip(int xPos, int yPos, int
       zPos, int wPos)
211 {
212     bool x = xPos == 1;
213     bool y = yPos == 1;
214     bool z = zPos == 1;
215     bool w = wPos == 1;
216
217     if (dim == 2)
218     {
219         if (mode == U)
220             return (x || y);
221         else if (mode == I)
222             return (x && y);
223         else
224             return false;
225     }
226     else if (dim == 3)
227     {
```

```
228         if (mode == UU)
229             return (x || y || z);
230         else if (mode == II)
231             return (x && y && z);
232         else if (mode == IU)
233             return ((x && y) || (x && z) || (y && z));
234         else
235             return false;
236     }
237     else if (dim == 4)
238     {
239         if (mode == UUU)
240             return (x || y || z || w);
241         else if (mode == III)
242             return (x && y && z && w);
243         else if (mode == IIU)
244             return ((x && y && z) ||
245                     (w && y && z) ||
246                     (x && w && z) ||
247                     (x && y && w));
248         else if (mode == UUI)
249             return ((x || y || z) &&
250                     (w || y || z) &&
251                     (x || w || z) &&
252                     (x || y || w));
253         else
254             return false;
255     }
256 }
257
258 template<int dim, int mode>
259 void cantorLevel<dim, mode>::draw(int recLevel, bool outline)
260 {
261     if (recLevel != rec)
262     {
263         if (!childGen)
264             generateChild();
265
266         child->draw(recLevel, outline);
267     }
268     else
269     {
270         if (!displayGen)
271             generateDisplay();
272
273         if (!outline)
274         {
275             ASSERT(glIsList(displayList));
```

```
276             glCallList(displayList);
277         }
278         else
279         {
280             ASSERT(glIsList(displayListOutline));
281             glCallList(displayListOutline);
282         }
283     }
284 }
285
286 template<int dim, int mode>
287 void cantorLevel<dim, mode>::generateDisplay()
288 {
289     if (displayGen)
290         return;
291
292     std::cerr << "Entering␣Generate␣Display" << std::endl;
293     std::cerr << "Recursion␣Level:␣" << rec << std::endl;
294
295     /**
296      * The following is a test, to see if I can get away
           without using adsoda spaces at all.
297      */
298     // Set up iterators
299     std::vector<Solid *>::iterator it = solids.begin();
300     std::vector<Solid *>::iterator end = solids.end();
301
302     // Have empty lights vector
303     std::vector<Light> lights;
304     lights.push_back(Light(dim, 1, 1, 1));
305     lights[0].Normalize();
306
307     // Set some ambient color we actually ignore
308     Color ambient = Color(.7, .7, .7);
309
310     // Start Display List
311
312     std::cerr << "Generating␣Display␣Lists:␣" << std::endl;
313     innerList = glGenLists(1);
314
315     glNewList(innerList, GL_COMPILE);
316
317     for (; it != end; it++)
318     {
319         if (dim == 4)
320         {
321             // Make a copy:
322             Solid * copy = new Solid(*(*it));
```

```
323             // Rotate:
324             // 4D rotation matrix :)
325             AMatrix rotationMatrix4D1(4, 4);
326             // rotationMatrix4D1.CreateRotationMatrix(1, 4,
                   3*PI/4);
327             rotationMatrix4D1.CreateRotationMatrix(1, 4, PI/4);
328
329             AMatrix rotationMatrix4D2(4, 4);
330             // rotationMatrix4D2.CreateRotationMatrix(2, 4,
                   2*PI/4);
331             rotationMatrix4D2.CreateRotationMatrix(2, 4, PI/4);
332
333             AMatrix rotationMatrix4D3(4, 4);
334             // rotationMatrix4D3.CreateRotationMatrix(2, 3,
                   1*PI/4);
335             rotationMatrix4D3.CreateRotationMatrix(3, 4, PI/4);
336
337             AMatrix rotationMatrix4D(4, 4);
338             rotationMatrix4D = rotationMatrix4D1 *
                   rotationMatrix4D2 * rotationMatrix4D3;
339
340             copy->Transform(rotationMatrix4D);
341
342             // Projecting:
343             std::vector<Solid *> projected;
344             copy->EnsureAdjacencies();
345             copy->Project(projected, lights, ambient);
346
347             // Ensure Adjacencies, render, and cleanup
348             std::vector<Solid *>::iterator curr =
                   projected.begin();
349             std::vector<Solid *>::iterator projEnd =
                   projected.end();
350             for (; curr != projEnd; curr++)
351             {
352                 (*curr)->EnsureAdjacencies();
353                 (*curr)->DrawUsingOpenGL3D(lights, ambient,
                       false, true);
354                 (*curr)->clearAdjacencies();
355                 delete *curr;
356                 *curr = NULL;
357             }
358             projected.clear();
359             delete copy;
360             copy = NULL;
361         }
362         else
363         {
```

```
364             // Ensure Adjacencies:
365             (*it)->EnsureAdjacencies();
366             // Render:
367             if (dim == 2)
368             {
369                 (*it)->DrawUsingOpenGL2D(false, true);
370                 // std::cerr << "Passed Draw 2D" << std::endl;
371             }
372             else
373             {
374                 (*it)->DrawUsingOpenGL3D(lights, ambient,
                        false, true);
375             }
376             // Cleanup:
377             (*it)->clearAdjacencies();
378         }
379         // std::cerr << "Passed clearAdjacencies" << std::endl;
380     }
381
382     glEndList();
383     displayGen = true;
384     lights.clear();
385
386     displayList = glGenLists(1);
387     glNewList(displayList, GL_COMPILE);
388     glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
389     glCallList(innerList);
390     glEndList();
391
392
393     displayListOutline = glGenLists(1);
394     glNewList(displayListOutline, GL_COMPILE);
395     glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
396     glCallList(innerList);
397     glEndList();
398
399     std::cerr << "DisplayList: " << displayList << std::endl;
400     std::cerr << "OutlineList: " << displayListOutline <<
        std::endl;
401
402     std::cerr << "Finished Rendering" << std::endl;
403 }
404
405 template<int dim, int mode>
406 void cantorLevel<dim, mode>::generateChild()
407 {
408     if (childGen)
409         return;
```

```
410     child = new cantorLevel(rec + 1, solids, sideLength/3);
411     childGen = true;
412     std::cerr << "Child Generated Successfully" << std::endl;
413     std::cerr << "Clearing Solids to Free up Space" <<
            std::endl;
414     if (!displayGen)
415         generateDisplay();
416
417     freeSolids();
418 }
419
420 template<int dim, int mode>
421 void cantorLevel<dim, mode>::generateRecLevel(int recL)
422 {
423     if (recL > rec)
424     {
425         if (!childGen)
426             generateChild();
427         child->generateRecLevel(recL);
428     }
429 }
430
431 #endif
```

2.2 adsoda_conversion

2.2.1 cc_adsoda.cpp

```
1  //*************************************************************
2  // Syzygy is licensed under the BSD license v2
3  // see the file SZG_CREDITS for details
4  //*************************************************************
5
6  // precompiled header include MUST appear as the first
       non-comment line
7  #include "arPrecompiled.h"
8  // MUST come before other szg includes. See
       arCallingConventions.h for details.
9  #define SZG_DO_NOT_EXPORT
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <iostream>
13 #include "arMasterSlaveFramework.h"
14 #include "arInteractableThing.h"
15 #include "arInteractionUtilities.h"
16 #include "arGlut.h"
17 #include "cc_adsoda_state.h"
18
19 using namespace std;
20
21 // Some global functions ...
22 void initState(State &state);
23 // void parseArgs(State &state, char *arg_string);
24 void parseArgs(State &state, int numArgs, const char * const
       *args);
25 void drawDemoFrame(State &state);
26 void prepareDemoFrame(State &state);
27 void drawDemoFrame(void);
   ...
32 // Unit conversions.  Tracker (and cube screen descriptions)
       use feet.
33 // Atlantis, for example, uses 1/2-millimeters, so the
       appropriate conversion
34 // factor is 12*2.54*20.
35 const float FEET_TO_LOCAL_UNITS = 1.;
36
37 // Near & far clipping planes.
38 const float nearClipDistance = .1*FEET_TO_LOCAL_UNITS;
39 const float farClipDistance = 100.*FEET_TO_LOCAL_UNITS;
   ...
129 class SkeletonFramework: public arMasterSlaveFramework
130 {
```

```
131 public:
132     SkeletonFramework();
133     bool onStart( arSZGClient& SZGClient );
134     void onWindowStartGL( arGUIWindowInfo* );
135     void onPreExchange( void );
136     void onPostExchange( void );
137 //    void onWindowInit( void );
138     void onDraw( arGraphicsWindow& win, arViewport& vp );
139 //    void onDisconnectDraw( void );
140 //    void onPlay( void );
141     void onWindowEvent( arGUIWindowInfo* );
142 //    void onCleanup( void );
143 //    void onUserMessage( const string& messageBody );
144 //    void onOverlay( void );
145 //    void onKey( unsigned char key, int x, int y );
146     void onKey( arGUIKeyInfo* );
147 //    void onMouse( arGUIMouseInfo* );
148 private:
    ...
160     State s_state;
161 };
162
163
164 SkeletonFramework::SkeletonFramework() :
       arMasterSlaveFramework()//, _squareHighlightedTransfer(0)
165 {
166     /* Nothing Here */
167     initState(s_state);
    ...
193     // Setup navigation, so we can drive around with the
           joystick
194     //
195     // Tilting the joystick by more than 20% along axis 1 (the
           vertical on ours) will cause
196     // translation along Z (forwards/backwards). This is
           actually the default behavior, so this
197     // line isn't necessary.
198     setNavTransCondition( 'z', AR_EVENT_AXIS, 1, 0.2 );
199
200     // Tilting joystick left or right will rotate left/right
           around vertical axis (default is left/right
201     // translation)
202     setNavRotCondition( 'y', AR_EVENT_AXIS, 0, 0.2 );
203
204     // Set translation & rotation speeds to 5 ft/sec & 30
           deg/sec (defaults)
205     setNavTransSpeed( 5. );
```

```
206     setNavRotSpeed( 30. );
207
208     // set square's initial position
209     // _square.setMatrix( ar_translationMatrix(0,5,-6) );
210
211     return true;
212 }
213
214 // Method to initialize each window (because now a Syzygy app
      can
215 // have more than one).
216 void SkeletonFramework::onWindowStartGL( arGUIWindowInfo* )
217 {
218     // OpenGL initialization
219     glClearColor(0,0,0,0);
220 }
221
222
223 // Method called before data is transferred from master to
      slaves. Only called
224 // on the master. This is where anything having to do with
225 // processing user input or random variables should happen.
226 void SkeletonFramework::onPreExchange()
227 {
228     // Do stuff on master before data is transmitted to slaves.
229
230     // handle joystick-based navigation (drive around). The
          resulting
231     // navigation matrix is automagically transferred to the
          slaves.
232     navUpdate();

    ...
245 }
246
247 // Method called after transfer of data from master to slaves.
      Mostly used to
248 // synchronize slaves with master based on transferred data.
249 void SkeletonFramework::onPostExchange()
250 {
251     // Do stuff after slaves got data and are again in sync
          with the master.
252     if (!getMaster())
253     {

    ...
261     }
262 }
263
```

```
264 void SkeletonFramework::onDraw( arGraphicsWindow& /*win*/,
      arViewport& /*vp*/ )
265 {
266     // Load the navigation matrix.
267     loadNavMatrix();
268     // Draw stuff.
269     // _square.draw();
270     // _effector.draw();
271 
272     glPushMatrix();
273     glMultMatrixf(ar_translationMatrix(0,6,-6).v);
274 
275     prepareDemoFrame(s_state);
276     drawDemoFrame(s_state);
277 
278     glPopMatrix();
279 
280 }
281 
282 // Catch key events.
283 void SkeletonFramework::onKey( arGUIKeyInfo* keyInfo )
284 {
285     if (!keyInfo)
286         return;
287     arGUIKey key = keyInfo->getKey();
288     cout << "Key␣ascii␣value␣=␣" << key << endl;
289     cout << "Key␣ctrl␣␣␣␣value␣=␣" << keyInfo->getCtrl() <<
          endl;
290     cout << "Key␣alt␣␣␣␣␣value␣=␣" << keyInfo->getAlt() <<
          endl;
291     string stateString;
292     arGUIState state = keyInfo->getState();
293     if (state == AR_KEY_DOWN)
294     {
295         stateString = "DOWN";
296         if (key == 115) //'s'
297             s_state.outlinePolys = !s_state.outlinePolys;
298     }
299     else if (state == AR_KEY_UP)
300     {
301         stateString = "UP";
302     }
303     else if (state == AR_KEY_REPEAT)
304     {
305         stateString = "REPEAT";
306     }
307     else
308     {
```

```
309         stateString = "UNKNOWN";
310     }
311     cout << "Key␣state␣=␣" << stateString << endl;
312 }
313
314 // This is how we have to catch reshape events now, still
315 // dealing with the fallout from the GLUT->arGUI conversion.
316 // Note that the behavior implemented below is the default.
317 void SkeletonFramework::onWindowEvent( arGUIWindowInfo*
       winInfo )
318 {
319     // The values are defined in src/graphics/arGUIDefines.h.
320     // arGUIWindowInfo is in arGUIInfo.h
321     // The window manager is in arGUIWindowManager.h
322     if (winInfo->getState() == AR_WINDOW_RESIZE)
323     {
324         const int windowID = winInfo->getWindowID();
325
326         #ifdef UNUSED
327             const int x = winInfo->getPosX();
328             const int y = winInfo->getPosY();
329         #endif
330
331         const int width = winInfo->getSizeX();
332         const int height = winInfo->getSizeY();
333         getWindowManager()->setWindowViewport( windowID, 0, 0,
              width, height );
334     }
335 }
336
337 int main(int argc, char** argv)
338 {
339     SkeletonFramework framework;
340     // Tell the framework what units we're using.
341     framework.setUnitConversion(FEET_TO_LOCAL_UNITS);
342     framework.setClipPlanes(nearClipDistance, farClipDistance);
343
344     if (!framework.init(argc, argv))
345     {
346         return 1;
347     }
348
349     // Never returns unless something goes wrong
350     return framework.start() ? 0 : 1;
351 }
```

2.2.2 cc_adsoda_demo.cpp

Here is where the "prepareDemoFrame(s_state)" and "drawDemoFrame(s_state)" called in cc_adsoda.cpp are.

```
1  #include "cc_adsoda_space.h"
2  #include "cc_adsoda_solid.h"
3  #include "cc_adsoda_light.h"
4  #include "cc_adsoda_amatrix.h"
5  #include "cc_adsoda_face.h"
6  #include "cc_adsoda_debug.h"
7  #include "cc_adsoda_state.h"
8  #include <stdlib.h>
9  #include <iostream>
10
11 //====  PROTOTYPES
12
13 Space *init2DDemo(void);
14 Space *init3DDemo(void);
15 Space *init4DDemo(void);
16
17 Space *Process1D(State &state, Space *space1D);
18 Space *Process2D(State &state, Space *space2D);
19 Space *Process3D(State &state, Space *space3D);
20 Space *Process4D(State &state, Space *space4D);
21
22 void drawcube(void);
23
24
25
26
27
28 void initState(State &state)
29 {
30
31   // Initialize the global state
32     state.outlinePolys = false;
33     state.fillPolys = true;
34     state.dimension = 4;
35     state.draw1D = false;
36     state.draw2D = false;
37     state.draw3D = false;
38     state.removeHidden2D = false;
39     state.removeHidden3D = false;
40     state.removeHidden4D = false;
41     state.rotate2D = false;
42     state.rotate3D = false;
43     state.rotate4D = false;
44     state.demoInitialized = false;
```

```
45      state.drawcubeFlag = false;
46
47      state.theta = 0;
48      state.rho = 0;
49      state.phi = 0;
50
51      // Fabs Code:
52      state.demoSpace = NULL;
53      state.draw1DSpace = NULL;
54      state.draw2DSpace = NULL;
55      state.draw3DSpace = NULL;
56      // End Fabs Code
57
58  }  //==== initState() ====//
59
60
61  // Initialize the demo
62  void initDemo(State &state)
63  {
64
65      if (state.dimension == 2)
66          state.demoSpace = init2DDemo();
67
68      else if (state.dimension == 3)
69          state.demoSpace = init3DDemo();
70
71      else if (state.dimension == 4)
72          state.demoSpace = init4DDemo();
73
74      else
75      {
76          std::cerr << "Error: there are only demos for
                state.dimensions 2, 3, and 4" << std::endl;
77          exit(-1);
78      }
79
80      state.demoInitialized = true;
81
82  }  //==== initDemo() ====
83
84
85
86  // Prepare one frame of the demo
87  void prepareDemoFrame(State &state)
88  {
89
90      // Clear out last frame's spaces
91      if (state.draw3DSpace)
```

```
92      {
93          delete state.draw3DSpace;
94          state.draw3DSpace = NULL;
95      }
96
97      if (state.draw2DSpace)
98      {
99          delete state.draw2DSpace;
100         state.draw2DSpace = NULL;
101     }
102
103     if (state.draw1DSpace)
104     {
105         delete state.draw1DSpace;
106         state.draw1DSpace = NULL;
107     }
108
109     // Initialize the demo, if we haven't yet
110     if (!state.demoInitialized)
111         initDemo(state);
112
113     Space *workingSpace = state.demoSpace;
114
115     if (workingSpace && (state.dimension >= 4) &&
           (workingSpace->Dimension() == 4))
116         workingSpace = Process4D(state, state.demoSpace);
117
118     Space *workingSpace3D = workingSpace;
119
120     if (workingSpace && (state.dimension >= 3) &&
           (workingSpace->Dimension() == 3))
121         workingSpace = Process3D(state, workingSpace);
122
123     Space *workingSpace2D = workingSpace;
124
125     if (workingSpace && (state.dimension >= 2) &&
           (workingSpace->Dimension() == 2))
126         workingSpace = Process2D(state, workingSpace);
127
128     Space *workingSpace1D = workingSpace;
129
130     if (workingSpace && (state.dimension >= 1) &&
           (workingSpace->Dimension() == 1))
131         workingSpace = Process1D(state, workingSpace);
132
133     if (workingSpace3D && (workingSpace3D != state.demoSpace))
134         delete workingSpace3D;
135
```

```
136     if (workingSpace2D && (workingSpace2D != state.demoSpace))
137         delete workingSpace2D;
138 
139     if (workingSpace1D && (workingSpace1D != state.demoSpace))
140         delete workingSpace1D;
141 
142 
143     // Make sure all the adjacencies of all the objects are
        ready,
144     // so we can draw without modifying the space object
145     if (state.draw3DSpace)
146         state.draw3DSpace->EnsureAdjacencies();
147 
148     if (state.draw2DSpace)
149         state.draw2DSpace->EnsureAdjacencies();
150 
151     if (state.draw1DSpace)
152         state.draw1DSpace->EnsureAdjacencies();
153 
154 }  //==== prepareDemoFrame() ====//
155 
156 
157 
158 // Display one frame of the demo
159 void drawDemoFrame(State &state)
160 {
161 
162     if (state.drawcubeFlag)
163         drawcube();
164 
165     if (state.draw3DSpace)
166         state.draw3DSpace->DrawUsingOpenGL3D(state.outlinePolys,
            state.fillPolys);
167 
168     if (state.draw2DSpace)
169         state.draw2DSpace->DrawUsingOpenGL2D(state.outlinePolys,
            state.fillPolys);
170 
171     if (state.draw1DSpace)
172         state.draw1DSpace->DrawUsingOpenGL1D(state.outlinePolys,
            state.fillPolys);
173 
174 }  //==== drawDemoFrame() ====//
175 
176 
177 
178 Space *Process1D(State &state, Space *space1D)
179 {
```

```
180
181   // If we're drawing, remember the space to draw
182     if (state.draw1D)
183         state.draw1DSpace = new Space(*space1D);
184     else
185         state.draw1DSpace = NULL;
186
187     return NULL;
188
189 }  //==== Process1D() ====//
190
191
192
193 Space *Process2D(State &state, Space *space2D)
194 {
195
196     // Copy the space if we're going to change it
197     Space *space2Dcopy = new Space(*space2D);
198
199     // Rotate the space if requested
200     if (state.rotate2D)
201     {
202         // Scale and rotate the space
203         Vector scaleVector(2);
204         scaleVector.coordinates[0] = 1;
205         scaleVector.coordinates[1] = 1;
206         AMatrix scaleMatrix2D(2, 2);
207         scaleMatrix2D.CreateScaleMatrix(scaleVector);
208         AMatrix rotationMatrix2D(2, 2);
209         rotationMatrix2D.CreateRotationMatrix(1, 2,
              state.theta);
210         AMatrix transformMatrix2D(2, 2);
211         transformMatrix2D = scaleMatrix2D * rotationMatrix2D;
212         space2Dcopy->Transform(transformMatrix2D);
213
214         state.theta += 0.01;
215
216     }  // if rotate
217
218     if (state.removeHidden2D)
219         space2Dcopy->RemoveHiddenSolids();
220
221     // Project to 1D, if we're going to draw it there
222     Space *space1D = NULL;
223     if (state.draw1D)
224     {
225         space1D = new Space(1, Color(0.1, 0.1, 0.1));
226         space2Dcopy->Project(space1D);
```

```
227     }
228
229     // If we're drawing, remember the space to draw; else,
           delete the copy space
230     if (state.draw2D)
231     {
232         state.draw2DSpace = space2Dcopy;
233     }
234     else
235     {
236         state.draw2DSpace = NULL;
237         delete space2Dcopy;
238     }
239
240     return space1D;
241
242 }  //==== Process2D() ====//
243
244
245
246 Space *Process3D(State &state, Space *space3D)
247 {
248
249     // Copy the space if we're going to change it
250     Space *space3Dcopy = new Space(*space3D);
251
252     // Rotate the space if requested
253     if (state.rotate3D)
254     {
255         // Rotate the space
256         AMatrix rotationMatrix3D(3, 3);
257         rotationMatrix3D.CreateRotationMatrix(2, 3,
               state.theta);
258         state.theta += -0.01;
259         AMatrix rotationMatrix3D2(3, 3);
260         rotationMatrix3D2.CreateRotationMatrix(1, 3,
               2*state.theta);
261         state.theta += -0.01;
262         space3Dcopy->Transform(rotationMatrix3D*rotationMatrix3D2);
263
264     }  // if rotate
265
266     // Remove hidden solids, if requested
267     if (state.removeHidden3D)
268         space3Dcopy->RemoveHiddenSolids();
269
270     // Project to 2D, if we're going to draw it there
271     Space *space2D = NULL;
```

```
272     if (state.draw2D)
273     {
274         space2D = new Space(2, Color(0.1, 0.1, 0.1));
275         space3Dcopy->Project(space2D);
276     }
277
278     // If we're drawing, remember the space to draw; else
           delete the copy space
279     if (state.draw3D)
280         state.draw3DSpace = space3Dcopy;
281     else
282     {
283         state.draw3DSpace = NULL;
284         delete space3Dcopy;
285     }
286
287     return space2D;
288
289 }  //==== Process3D() ====//
290
291
292
293 Space *Process4D(State &state, Space *space4D)
294 {
295
296     Space *space4Dcopy;
297
298     // Copy the space if we're going to change it
299     if (state.rotate4D || state.removeHidden4D)
300         space4Dcopy = new Space(*space4D);
301     else
302         space4Dcopy = space4D;
303
304     // Rotate the space if requested
305     if (state.rotate4D)
306     {
307        // Rotate the copy
308         AMatrix rotationMatrix4D1(4, 4);
309         AMatrix rotationMatrix4D2(4, 4);
310         AMatrix rotationMatrix4D3(4, 4);
311         AMatrix rotationMatrix4D(4, 4);
312         rotationMatrix4D1.CreateRotationMatrix(1, 4,
              2*state.rho);
313         rotationMatrix4D2.CreateRotationMatrix(2, 4,
              2*state.theta);
314         rotationMatrix4D3.CreateRotationMatrix(3, 2,
              2*state.phi);
```

```
315         rotationMatrix4D = rotationMatrix4D1 *
               rotationMatrix4D2 * rotationMatrix4D3;
316         space4Dcopy->Transform(rotationMatrix4D);
317
318         // std::cout << "Theta: " << state.theta << std::endl;
319         // std::cout << "Rho: " << state.rho << std::endl;
320         // std::cout << "Phi: " << state.phi << std::endl;
321         state.theta += 0.001;
322         // state.rho += 0.001;
323         // state.phi += 0.001;
324
325     }  // if rotate
326
327     if (state.removeHidden4D)
328         space4Dcopy->RemoveHiddenSolids();
329
330     // Project to 3D, if we're going to draw it there
331     Space *space3D = NULL;
332     if (state.draw3D)
333     {
334         space3D = new Space(3, Color(0.7, 0.7, 0.7));
335         // Light *light = new Light(3, 0.8, 0.8, 0.8);
336         // light->coordinates[0] = -100;
337         // light->coordinates[1] = -100;
338         // light->coordinates[2] = -100;
339         // space3D->AddLight(light);
340         space4Dcopy->Project(space3D);
341     }
342
343     if (space4Dcopy != space4D)
344         delete space4Dcopy;
345
346     // state.theta += -0.01;
347
348     return space3D;
349
350 }  //==== Process4D() ====//
```

Fin.