

ภาคผนวก ข

โปรแกรมการปูลายผิวตามค่าพลาศาสตร์ของไหล

การพัฒนาโปรแกรมสำหรับใช้ทดสอบการทำงานของกรปูลายผิวตามค่าพลาศาสตร์ของไหลนี้ ได้ประยุกต์ใช้โปรแกรมต้นแบบจากหนังสือ “GPU Gems2 Programming Techniques for High-Performance Graphics and General-Purpose Computation” ของสำนักพิมพ์ Addison-Wesley โดยเฉพาะในบทที่ 12 “Tile-Based Texture Mapping” ซึ่งโปรแกรมสามารถหาได้จาก http://http.download.nvidia.com/developer/GPU_Gems_2/CD/Content/12.zip มาเป็นต้นแบบโปรแกรม สำหรับใช้ในส่วนของการสังเคราะห์ และวิธีการปูลายผิววงทอิล โดยทำการปรับปรุงโปรแกรม ให้สามารถแสดงผลพัทธ์ของลายผิวตามค่าพลาศาสตร์ของไหล ด้วยการแก้ไขโปรแกรม TileMapping.cpp และ CgWangTileT.cpp

การปรับปรุงโปรแกรม TileMapping.cpp

```
1 /*
2 * TileMapping.cpp
3 *
4 * Li-Yi Wei
5 * 8/9/2003
6 *
7 * Modify by Mr.Supachai 2010 *
8 */
9
10 #pragma warning (disable: 4786)
11
12 #ifdef WIN32
13 #include <windows.h>
14 #endif
15
16 #include <GL/gl.h>
17 #include <GL/glut.h>
18 #include <GL/glext.h>
19
20 #include <iostream>
21 #include <fstream>
22 #include <sstream>
23 #include <iomanip>
24 #include <vector>
25 #include <deque>
26
27 #include <math.h>
28 #include <ctime>
29 #include <time.h>
30 #include <assert.h>
31 using namespace std;
32
33 #include "WangTiles.hpp"
```

```

34 #include "WangTilesProcessor.hpp"
35 #include "CgWangTilesT.hpp"
36 #include "CgWangTilesC.hpp"
37
38 #include "Timer.hpp"
39 #include "EventTimer.hpp"
40 #include "FrameBuffer.hpp"
41
42 #include <glh/glh_extensions.h>
43
44 #define REQUIRED_EXTENSIONS "WGL_ARB_pbuffer " \
45 "WGL_ARB_pixel_format " \
46 "GL_NV_float_buffer " \
47 "GL_ARB_multitexture "
48
49 class ExitException : public Exception
50 {
51     public:
52     ExitException(const string & message);
53 };
54
55 ExitException::ExitException(const string & message) : Exception(message)
56 {
57     // nothing to do
58 }
59
60 CgWangTilesT * pCgWangTilesT = 0;
61 CgWangTilesC * pCgWangTilesC = 0;
62
63 class P4
64 {
65     public:
66     float x, y, z, w;
67 };
68
69 class TileCache
70 {
71     public:
72     TileCache(void) : _cacheLines(0) {};
73     ~TileCache(void) {};
74
75     int Get(const WangTiles::Tile & tile, vector< vector<P4> > & image) const
76     {
77         int index = Index(tile);
78
79         if(index >= 0)
80         {
81             image = _cacheLines[index].image;
82             return 1;
83         }
84         else
85         {
86             return 0;
87         }
88     };
89
90     int Put(const WangTiles::Tile & tile, const vector< vector<P4> > & image)
91     {
92         int index = Index(tile);
93
94         if(index < 0)
95         {
96             Entry newGuy;

```

```

97         newGuy.tile = tile;
98         _cacheLines.push_back(newGuy);
99         index = _cacheLines.size() - 1;
100     }
101
102     assert(index >= 0);
103
104     _cacheLines[index].image = image;
105     return 1;
106 };
107
108 int TileHeight(void) const
109 {
110     if(_cacheLines.size() > 0)
111     {
112         return _cacheLines[0].image.size();
113     }
114     else
115     {
116         return 0;
117     }
118 };
119
120 int TileWidth(void) const
121 {
122     if(_cacheLines.size() > 0)
123     {
124         return _cacheLines[0].image[0].size();
125     }
126     else
127     {
128         return 0;
129     }
130 };
131
132 protected:
133 int Index(const WangTiles::Tile & tile) const
134 {
135     int result = -1;
136     for(unsigned int i = 0; i < _cacheLines.size(); i++)
137     {
138         if(tile.ID() == _cacheLines[i].tile.ID())
139         {
140             result = i;
141         }
142     }
143
144     return result;
145 };
146
147 struct Entry
148 {
149     WangTiles::Tile tile;
150     vector< vector<P4> > image;
151 };
152
153 deque<Entry> _cacheLines;
154 };
155
156 struct CameraCoord
157 {
158     CameraCoord(void)

```

```

159     {
160         from[0] = from[1] = from[2] = from[3] = 0;
161         to[0] = to[1] = to[2] = to[3] = 0;
162     }
163
164     CameraCoord(const float from0,
165     const float from1,
166     const float from2,
167     const float from3,
168     const float to0,
169     const float to1,
170     const float to2,
171     const float to3)
172 {
173     from[0] = from0; to[0] = to0;
174     from[1] = from1; to[1] = to1;
175     from[2] = from2; to[2] = to2;
176     from[3] = from3; to[3] = to3;
177 }
178
179 float from[4], to[4];
180 };
181
182 class GlobalParameters
183 {
184     public:
185     int resultWindow, tileWindow, cornerWindow, timerWindow;
186
187     GLuint tilesTextureID;
188     GLuint cornersTextureID;
189     GLuint tileMappingTextureID;
190     GLuint cornerMappingTextureID;
191     GLuint resultTextureID;
192     GLuint permutationTextureID;
193
194     int numHColors;
195     int numVColors;
196     int numTilesPerColor;
197
198     int tileSize;
199
200     int mappingTextureHeight;
201     int mappingTextureWidth;
202
203     int permutationTextureSize;
204
205     int reset;
206
207     int compaction; // 0 for random, 1 for even
208
209     float cornerSharpness;
210
211     Timer timer;
212     int eventHistory;
213     EventTimer *pEventTimer;
214
215     float startTime, endTime;
216     int numFrames;
217
218     WangTiles::TileSet *pTileSet;
219
220     TileCache tileCache;

```

```

221     int changeTileCache;
222
223     vector< vector<WangTiles::Tile> > tileCompaction;
224     vector< vector<WangTiles::Tile> > cornerCompaction;
225
226     vector< vector<WangTiles::Tile> > tileMapping;
227     vector< vector<WangTiles::Tile> > cornerMapping;
228
229     // scene options
230     int perspectiveView;// 1 or 0 control signal
231     int winHeight, winWidth;
232     float fov, eye[3], center[3];
233
234     // visualization path
235     deque<CameraCoord> cameraPath;
236     int cameraPathIndex;
237     int dumpAnimation;
238
239     int sanitize;
240 };
241
242 GlobalParameters global;
243
244 void AddLinearPath(const int numFrames,
245 const CameraCoord & start,
246 const CameraCoord & end,
247 deque<CameraCoord> & path)
248 {
249     CameraCoord delta;
250 {
251 for(int i = 0;i < 4;i++)
252 {
253     delta.from[i] = (end.from[i] - start.from[i])/(numFrames-1);
254     delta.to[i] = (end.to[i] - start.to[i])/(numFrames-1);
255 }
256 }
257
258 CameraCoord current = start;
259
260 for(int i = 0;i < numFrames;i++)
261 {
262     path.push_back(current);
263
264     for(int j = 0;j < 4;j++)
265     {
266         current.from[j] += delta.from[j];
267         current.to[j] += delta.to[j];
268     }
269 }
270 }
271
272 void BuildVisualizationPath(void)
273 {
274 // parameters
275     const int numZoomFrames = 100;
276     const int numPitchFrames = 50;
277     const int numMoveFrames = 200;
278
279     const float zFar = 2.0;
280     const float zNear = 0.03;
281     const float yShift = -0.15;
282     const float yPitch = 0.06;

```

```

283     const float yPitch1 = 0.32;
284     const CameraCoord zoomOut(0, yShift, zFar, 0, 0, yShift, 0, 0);
285     const CameraCoord zoomIn(0, yShift, zNear, 0, 0, yShift, 0, 0);
286     const CameraCoord p1(0, yShift, zNear, 0, 0, yPitch+yShift, 0, 0);
287     const CameraCoord p2(0, yPitch1+yShift, zNear, 0, 0,
yPitch+yPitch1+yShift, 0, 0)
;
288
289 // reset
290     global.cameraPath.clear();
291     global.cameraPathIndex = 0;
292
293 // zoom in
294     AddLinearPath(numZoomFrames, zoomOut, zoomIn, global.cameraPath);
295 // pitch up
296     AddLinearPath(numPitchFrames, zoomIn, p1, global.cameraPath);
297 // forward
298     AddLinearPath(numMoveFrames, p1, p2, global.cameraPath);
299
300     if(!global.dumpAnimation)
301     {
302         // backward
303             AddLinearPath(numMoveFrames, p2, p1, global.cameraPath);
304         // pitch down
305             AddLinearPath(numPitchFrames, p1, zoomIn, global.cameraPath);
306         // zoom out
307             AddLinearPath(numZoomFrames, zoomIn, zoomOut,
global.cameraPath);
308     }
309 }
310
311 void RenderQuad(const float centerX, const float centerY)
312 {
313     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
314
315     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
316     glBegin(GL_QUADS);
317
318     glColor3f(0.0, 0.0, 1.0); // blue
319     glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 0, 0);
320     glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 0, 0);
321     glVertex2f(centerX - 1, centerY - 1);
322
323     glColor3f(0.0, 1.0, 0.0); // green
324     glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1, 0);
325     glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1, 0);
326     glVertex2f(centerX + 1, centerY - 1);
327
328     glColor3f(1.0, 0.0, 0.0); // red
329     glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1, 1);
330     glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1, 1);
331     glVertex2f(centerX + 1, centerY + 1);
332
333     glColor3f(1.0, 1.0, 0.0); // yellow
334     glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 0, 1);
335     glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 0, 1);
336     glVertex2f(centerX - 1, centerY + 1);
337
338     glEnd();
339 }
340
341 void Enable(void)

```

```

342 {
343     if(pCgWangTilesT) pCgWangTilesT->Enable();
344     if(pCgWangTilesC) pCgWangTilesC->Enable();
345 }
346
347 void Disable(void)
348 {
349     if(pCgWangTilesT) pCgWangTilesT->Disable();
350     if(pCgWangTilesC) pCgWangTilesC->Disable();
351 }
352
353 void Display(void)
354 {
355     if( (glutGetWindow() == global.resultWindow) &&
356         (global.resultTextureID <= 0) )
357     {
358         if(global.tileMappingTextureID > 0)
359         {
360             glActiveTextureARB(GL_TEXTURE0_ARB);
361             glBindTexture(GL_TEXTURE_RECTANGLE_NV, global.tileMappingTextureID);
362             glEnable(GL_TEXTURE_RECTANGLE_NV);
363         }
364
365         if(global.cornerMappingTextureID > 0)
366         {
367             glActiveTextureARB(GL_TEXTURE1_ARB);
368             glBindTexture(GL_TEXTURE_RECTANGLE_NV, global.cornerMappingTextureID);
369             glEnable(GL_TEXTURE_RECTANGLE_NV);
370         }
371
372         Enable();
373     }
374     else // tileWindow or cornerWindow or timerWindow
375     {
376         Disable();
377     }
378
379     if(global.tilesTextureID > 0)
380     {
381         glActiveTextureARB(GL_TEXTURE0_ARB);
382         glBindTexture(GL_TEXTURE_2D, global.tilesTextureID);
383         if(glutGetWindow() != global.cornerWindow)
384         {
385             glEnable(GL_TEXTURE_2D);
386         }
387     }
388
389     if(global.cornersTextureID > 0)
390     {
391         glActiveTextureARB(GL_TEXTURE1_ARB);
392         glBindTexture(GL_TEXTURE_2D, global.cornersTextureID);
393         if(glutGetWindow() != global.tileWindow)
394         {
395             glEnable(GL_TEXTURE_2D);
396         }
397     }
398
399     if(global.resultTextureID > 0)
400     {
401         if(glutGetWindow() == global.resultWindow)
402         {
403             glActiveTextureARB(GL_TEXTURE1_ARB);
404             glBindTexture(GL_TEXTURE_2D, global.cornersTextureID);

```

```

405     glDisable(GL_TEXTURE_2D);
406
407     glActiveTextureARB(GL_TEXTURE0_ARB);
408     glBindTexture(GL_TEXTURE_2D, global.resultTextureID);
409     glEnable(GL_TEXTURE_2D);
410     }
411 }
412
413 if((global.cameraPath.size()) > 0 && (glutGetWindow() ==
global.resultWindow))
414 {
415     // projection
416     glMatrixMode(GL_PROJECTION);
417     glLoadIdentity();
418     gluPerspective(global.fov,
419     1.0*global.winWidth/global.winHeight,
420     0.01, 100.0);
421
422     // lookat
423     glMatrixMode(GL_MODELVIEW);
424
425     glPopMatrix();
426     glPushMatrix();
427
428     const CameraCoord & camera =
global.cameraPath[global.cameraPathIndex];
429     global.cameraPathIndex =
(global.cameraPathIndex+1)%global.cameraPath.size();
430
431     if(global.dumpAnimation && (global.cameraPathIndex == 0))
432     {
433         throw ExitException("done animation dumping");
434     }
435
436     gluLookAt(camera.from[0], camera.from[1], camera.from[2],
437     camera.to[0],
438     camera.to[1],
439     camera.to[2],
440     0, 1, 0);
441
442     glTranslatef(0.0, 0.0, -camera.from[2]);
443 }
444
445 if(global.perspectiveView && (glutGetWindow() == global.resultWindow))
446 {
447     // projection
448     glMatrixMode(GL_PROJECTION);
449     glLoadIdentity();
450     gluPerspective(global.fov,
451     1.0*global.winWidth/global.winHeight,
452     0.1, 10.0);
453
454     // zoom
455     glMatrixMode(GL_MODELVIEW);
456
457     glPopMatrix();
458     glPushMatrix();
459
460     gluLookAt(global.eye[0], global.eye[1], global.eye[2],
461     global.center[0],
462     global.center[1],
463     global.center[2],

```

```

464     0, 1, 0);
465
466     glTranslatef(0.0, 0.0, -global.eye[2]);
467 }
468
469 RenderQuad(0, 0);
470
471 if(global.pEventTimer && (glutGetWindow() == global.resultWindow))
472 {
473     global.pEventTimer->RecordTime(global.timer.CurrentTime());
474 }
475
476 if(global.startTime <= 0)
477 {
478     global.startTime = global.timer.CurrentTime();
479 }
480
481 if(glutGetWindow() == global.resultWindow)
482 {
483     global.numFrames++;
484 }
485
486 if(global.dumpAnimation)
487 {
488     glReadBuffer(GL_BACK);
489     glFlush();
490
491     int params[6];
492     glGetIntegerv(GL_VIEWPORT, params);
493     glutSetCursor(GLUT_CURSOR_WAIT);
494     char filename[256];
495     sprintf(filename, "frame_%.4d.ppm", global.cameraPathIndex);
496     FrameBuffer::WriteColor(params[2], params[3], filename);
497     glutSetCursor(GLUT_CURSOR_INHERIT);
498 }
499
500 glutSwapBuffers();
501 }
502
503 void DisplayString(GLfloat x, GLfloat y, GLfloat scale, const string &
message)
504 {
505     glPushMatrix();
506
507     glTranslatef(x, y, 0);
508     glScalef(scale, scale, scale);
509
510     for(int i = 0; i < message.length(); i++)
511         glutStrokeCharacter(GLUT_STROKE_ROMAN, message[i]);
512
513     glPopMatrix();
514 }
515
516 void TimerDisplay(void)
517 {
518     if(global.pEventTimer)
519     {
520         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
521
522         stringstream strResult;
523
524         float frameRate = 0;

```

```

525     float elapsedTime = global.pEventTimer->ElapsedTime(-1,
global.eventHistory);
526
527     if(elapsedTime > 0)
528     {
529         frameRate = global.eventHistory/elapsedTime;
530     }
531
532     strResult << frameRate;
533
534     string message(strResult.str(), strResult.pcount());
535     strResult.rdbuf()->freeze(0);
536
537     glColor3f(1.0, 1.0, 1.0);
538     DisplayString(-0.75, 0, 0.002, message);
539
540     glutSwapBuffers();
541 }
542 }
543
544 // colors contain 5 components:
545 // colors for bottom, right, top, and left edges,
546 // and central color
547 int CreateDiamondTile(const vector<P4> & colors,
548 const int height, const int width,
549 vector< vector<P4> > & result)
550 {
551     if(colors.size() < 5)
552     {
553         return 0;
554     }
555
556     if(result.size() != height)
557     {
558         result = vector< vector<P4> >(height);
559     }
560
561 {
562 for(int i = 0;i < result.size();i++)
563 {
564     if(result[i].size() != width)
565     {
566         result[i] = vector<P4>(width);
567     }
568 }
569 }
570
571 {
572 for(int i = 0;i < height;i++)
573 for(int j = 0;j < width;j++)
574 {
575     int eq1 = height * j - width * i;
576     int eq2 = height * j + width * i - height*width;
577
578     P4 color;
579
580     if( (i > height/16) && (i < 15*height/16) &&
581 (j > width/16) && (j < 15*width/16) )
582         color = colors[4];
583     else if((eq1 > 0) && (eq2 <= 0)) color = colors[0];
584     else if((eq1 > 0) && (eq2 > 0)) color = colors[1];
585     else if((eq1 <= 0) && (eq2 > 0)) color = colors[2];

```

```

586     else color = colors[3];
587
588     result[i][j] = color;
589 }
590 }
591
592 // done
593 return 1;
594 }
595
596 void Randomize(vector<P4> & colors)
597 {
598     float tmp[5][4] = { {1.0,0.2,0.2,0.0},
599                       {0.2,1.0,0.2,0.0},
600                       {0.2,0.2,1.0,0.0},
601                       {0.8,0.8,0.2,0.0},
602                       {0.2,0.8,0.8,0.0}};
603     for(int i = 0; i < colors.size(); i++)
604     {
605         colors[i].x = tmp[i%5][0];
606         colors[i].y = tmp[i%5][1];
607         colors[i].z = tmp[i%5][2];
608         colors[i].w = tmp[i%5][3];
609     }
610 }
611
612 int TileCompaction(vector< vector<WangTiles::Tile> > & result,
613 const int compaction_method)
614 {
615     int status;
616
617     if(! global.pTileSet)
618     {
619         return 0;
620     }
621
622     if(compaction_method == 1)
623     {
624         status = WangTiles::OrthogonalCompaction(*global.pTileSet, result);
625     }
626     else
627     {
628         status = WangTiles::RandomCompaction(*global.pTileSet, result);
629     }
630
631     return status;
632 }
633
634 int CornerCompaction(vector< vector<WangTiles::Tile> > & result)
635 {
636     int status;
637
638     status = WangTiles::OrthogonalCornerCompaction(*global.pTileSet,
639 result);
640
641     return status;
642 }
643
644 int CreateDiamondTileSet(const int numHColors,
645 const int numVColors,
646 const int numTilesPerColor,
647 const int tileSize,
648 TileCache & cache)

```

```

648 {
649     if(global.tileCompaction.size() <= 0)
650     {
651         return 0;
652     }
653
654     const int tileTextureHeight = global.tileCompaction.size();
655     const int tileTextureWidth = global.tileCompaction[0].size();
656
657     // random tile colors
658     vector<P4> hColors(numHColors);
659     vector<P4> vColors(numVColors);
660     vector<P4> cColors(tileTextureHeight*tileTextureWidth);
661
662     Randomize(hColors);
663     Randomize(vColors);
664
665     for(int i=0;i<cColors.size();i++){
666         float temp = rand()*1.0/RAND_MAX;
667         cColors[i].w = temp;
668         cColors[i].x = 0.9;
669         cColors[i].y = 0.9;
670         cColors[i].z = 0.9;
671     }
672
673 {
674 for(int i = 0;i < tileTextureHeight;i++)
675 for(int j = 0;j < tileTextureWidth;j++)
676 {
677     // build the tile
678     vector<P4> colors(5);
679
680     WangTiles::Tile compactionTile = global.tileCompaction[i][j];
681
682     colors[0] = hColors[compactionTile.e0()];
683     colors[1] = vColors[compactionTile.e1()];
684     colors[2] = hColors[compactionTile.e2()];
685     colors[3] = vColors[compactionTile.e3()];
686     colors[4] = cColors[compactionTile.ID()];
687
688     vector< vector<P4> > tile;
689
690     if(! CreateDiamondTile(colors, tileSize, tileSize, tile))
691     {
692         throw Exception("Error in CreateDiamondTile");
693     }
694
695     cache.Put(compactionTile, tile);
696 }
697 }
698
699 return 1;
700 }
701
702 int CreateInputTileSet(const vector< vector<FrameBuffer::P3> > &
inputPacking,
703     const int maximumValue,
704     const int numHColors,
705     const int numVColors,
706     const int numTilesPerColor,
707     TileCache & cache)
708 {

```

```

709     if(global.tileCompaction.size() <= 0)
710     {
711         return 0;
712     }
713
714     const int packingTextureHeight = inputPacking.size();
715     const int packingTextureWidth = packingTextureHeight > 0 ?
inputPacking[0].size(): 0;
716     const int tileTextureHeight = global.tileCompaction.size();
717     const int tileTextureWidth = global.tileCompaction[0].size();
718     const int tileHeight = packingTextureHeight/tileTextureHeight;
719     const int tileWidth = packingTextureWidth/tileTextureWidth;
720
721     assert(packingTextureHeight%tileTextureHeight == 0);
722     assert(packingTextureWidth%tileTextureWidth == 0);
723
724 {
725     vector< vector<P4> > tile(tileHeight);
726     {
727         for(int i = 0;i < tileHeight;i++)
728         {
729             tile[i] = vector<P4>(tileWidth);
730         }
731     }
732
733     vector< vector<WangTiles::Tile> > goodTileCompaction;
734
735     if(! TileCompaction(goodTileCompaction, 1))
736     {
737         return 0;
738     }
739
740     for(int i = 0;i < tileTextureHeight;i++)
741     for(int j = 0;j < tileTextureWidth;j++)
742     {
743         // build the tile from good compaction
744         WangTiles::Tile compactionTile = goodTileCompaction[i][j];
745
746         // assign to the tile texture
747         for(int m = i*tileHeight;m < (i+1)*tileHeight;m++)
748         for(int n = j*tileWidth;n < (j+1)*tileWidth;n++)
749         {
750             FrameBuffer::P3 inputColor = inputPacking[m][n];
751
752             P4 outputColor;
753             outputColor.x = inputColor.r*1.0/maximumValue;
754             outputColor.y = inputColor.g*1.0/maximumValue;
755             outputColor.z = inputColor.b*1.0/maximumValue;
756             outputColor.w = 1.0;
757
758             tile[m - i*tileHeight][n - j*tileWidth] = outputColor;
759         }
760
761         cache.Put(compactionTile, tile);
762     }
763 }
764
765 return 1;
766 }
767
768 struct ImageSize
769 {

```

```

770     int height, width;
771 };
772
773 // return number of mipmap levels if successful, 0 else
774 // assuming RGBA pixel format, and data is of sufficient size
775 int SanitizeTileTexture(const vector< vector<WangTiles::Tile> > & tiles,
776 const int height, const int width,
777 float * data)
778 {
779 // error checking
780     if((height <= 0) || (width <= 0))
781     {
782         return 0;
783     }
784
785     // initialization
786     Array2D<int> pyramidSpec;
787     {
788         vector<ImageSize> sizeSpec;
789         ImageSize current;
790         current.height = height; current.width = width;
791
792         do
793         {
794             sizeSpec.push_back(current);
795
796             current.height /= 2; current.width /= 2;
797         }
798         while((current.height > 1) || (current.width > 1));
799
800         pyramidSpec = Array2D<int>(sizeSpec.size(), 3);
801         for(unsigned int i = 0; i < sizeSpec.size(); i++)
802         {
803             pyramidSpec[i][0] = sizeSpec[i].height;
804             pyramidSpec[i][1] = sizeSpec[i].width;
805             // assuming ARGB pixel format
806             pyramidSpec[i][2] = 4;
807         }
808     }
809
810     ImagePyramid<float> pyramid(pyramidSpec);
811
812     {
813         Array3D<float> & image = pyramid[0];
814
815         int index = 0;
816         for(int row = 0; row < image.Size(0); row++)
817             for(int col = 0; col < image.Size(1); col++)
818                 for(int cha = 0; cha < image.Size(2); cha++)
819                 {
820                     image[row][col][cha] = data[index++];
821                 }
822     }
823
824     // sanitize
825     if(! WangTilesProcessor::BoxSanitize(tiles, pyramid, pyramid,
826 global.sanitize ==
827 2))
828     {
829         return 0;
830     }
831 }

```

```

830 {
831     const int mismatch_level = WangTilesProcessor::TileMismatch(tiles,
pyramid);
832     if(mismatch_level >= 0)
833     {
834         cerr << "tile mismatch at level " << mismatch_level << endl;
835     }
836 }
837
838 // copy to output
839 {
840 int index = 0;
841
842 for(int level = 0; level < pyramid.NumLevels(); level++)
843 {
844     Array3D<float> & image = pyramid[level];
845
846     for(int row = 0; row < image.Size(0); row++)
847     for(int col = 0; col < image.Size(1); col++)
848     for(int cha = 0; cha < image.Size(2); cha++)
849     {
850         data[index++] = image[row][col][cha];
851     }
852 }
853 }
854
855 // done
856 return pyramid.NumLevels();
857 }
858
859 int CreateTileTexture(const GLuint tileTextureID,
860 const GLuint cornerTextureID,
861 const int numHColors,
862 const int numVColors,
863 const int numTilesPerColor,
864 const int tileSize)
865 {
866     // create the diamond tile set
867     if(global.changeTileCache &&
868        !CreateDiamondTileSet(numHColors, numVColors,
869        numTilesPerColor, tileSize,
870        global.tileCache))
871     {
872         throw Exception("error in CreateDiamondTileSet()");
873     }
874
875     if( (global.tileCompaction.size() <= 0) ||
876        (global.cornerCompaction.size() <= 0) )
877     {
878         return 0;
879     }
880
881     const int tileTextureHeight = global.tileCompaction.size();
882     const int tileTextureWidth = global.tileCompaction[0].size();
883
884     const int cornerTextureHeight = global.cornerCompaction.size();
885     const int cornerTextureWidth = global.cornerCompaction[0].size();
886
887     // Fill in the tile texture map.
888     const int glTileTextureHeight = tileSize * tileTextureHeight;
889     const int glTileTextureWidth = tileSize * tileTextureWidth;
890     float *data1 = new float[glTileTextureHeight*glTileTextureWidth*4*2];
891

```

```

892     {
893     for(int i = 0;i < tileTextureHeight;i++)
894     for(int j = 0;j < tileTextureWidth;j++)
895     {
896     WangTiles::Tile compactionTile = global.tileCompaction[i][j];
897
898     vector< vector<P4> > tile;
899
900     if(! global.tileCache.Get(compactionTile, tile))
901     {
902         throw Exception("Error in tileCache.Get()");
903     }
904
905     // assign to the tile texture
906     for(int m = i*tileSize;m < (i+1)*tileSize;m++)
907     for(int n = j*tileSize;n < (j+1)*tileSize;n++)
908     {
909         int op = m*glTileTextureWidth + n;
910         P4 color = tile[m - i*tileSize][n - j*tileSize];
911
912         data1[4*op + 0] = color.x;
913         data1[4*op + 1] = color.y;
914         data1[4*op + 2] = color.z;
915         data1[4*op + 3] = color.w;
916     }
917 }
918 }
919
920 // Fill in the corner texture map.
921 const int glCornerTextureHeight = tileSize *cornerTextureHeight;
922 const int glCornerTextureWidth = tileSize * cornerTextureWidth;
923 float *data2 = new float[glCornerTextureHeight*glCornerTextureWidth*4*2];
924
925 if(cornerTextureID > 0)
926 {
927     for(int i = 0;i < cornerTextureHeight;i++)
928     for(int j = 0;j < cornerTextureWidth;j++)
929     {
930         // build the tile
931         vector<P4> colors(5);
932
933         WangTiles::Tile compactionTile = global.cornerCompaction[i][j];
934
935         vector< vector<P4> > tile;
936
937         if(! global.tileCache.Get(compactionTile, tile))
938         {
939             throw Exception("Error in tileCache.Get()");
940         }
941
942         // assign to the tile texture
943         for(int m = i*tileSize;m < (i+1)*tileSize;m++)
944         for(int n = j*tileSize;n < (j+1)*tileSize;n++)
945         {
946             int op = m*glCornerTextureWidth + n;
947             P4 color = tile[m - i*tileSize][n - j*tileSize];
948
949             data2[4*op + 0] = color.x;
950             data2[4*op + 1] = color.y;
951             data2[4*op + 2] = color.z;
952             data2[4*op + 3] = color.w;
953         }

```

```

954     }
955 }
956
957 // valid window IDs
958 vector<int> validWins;
959
960 if(global.resultWindow) validWins.push_back(global.resultWindow);
961 if(global.tileWindow) validWins.push_back(global.tileWindow);
962 if(global.cornerWindow) validWins.push_back(global.cornerWindow);
963
964 int originalWin = glutGetWindow();
965
966 for(int i = 0; i < validWins.size(); i++)
967 {
968     glutSetWindow(validWins[i]);
969
970     // tile texture
971     glActiveTextureARB(GL_TEXTURE0_ARB);
972     glBindTexture(GL_TEXTURE_2D, tileTextureID);
973     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
974
975     glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP_SGIS, GL_TRUE);
976     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
977     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
978     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
979     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
980
981     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, glTileTextureWidth,
glTileTextureHeight, 0, GL_RGBA, GL_FLOAT, data1);
982
983     if(global.sanitize)
984     {
985         const int num_levels = SanitizeTileTexture(global.tileCompaction,
glTileTextureHeight, glTileTextureWidth, data1);
986
987         if(num_levels <= 0)
988         {
989             throw Exception("error in SanitizeTileTexture!");
990         }
991         return 0;
992     }
993
994     int offset = 0;
995     int glTileTextureMipWidth = glTileTextureWidth;
996     int glTileTextureMipHeight = glTileTextureHeight;
997
998     for(int i = 0; i < num_levels; i++, offset += glTileTextureMipWidth*
glTileTextureMipHeight*4, glTileTextureMipWidth/= 2,
glTileTextureMipHeight/= 2)
999     {
1000         glTexImage2D(GL_TEXTURE_2D, i, GL_RGBA, glTileTextureMipWidth,
glTileTextureMipHeight, 0, GL_RGBA, GL_FLOAT, &data1[offset]);
1001     }
1002
1003     // corner texture
1004     if(cornerTextureID > 0)
1005     {
1006         glActiveTextureARB(GL_TEXTURE1_ARB);
1007         glBindTexture(GL_TEXTURE_2D, cornerTextureID);
1008         glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
1009     }

```

```

1010  glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP_SGIS, GL_TRUE);
1011  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
1012  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
1013  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
1014  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
1015
1016  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, glCornerTextureWidth,
glCornerTextureHeight, 0, GL_RGBA, GL_FLOAT, data2);
1017
1018  if(global.sanitize)
1019  {
1020  const int num_levels = SanitizeTileTexture(global.cornerCompaction,
glTileTextureHeight, glTileTextureWidth, data2);
1021
1022  if(num_levels <= 0)
1023  {
1024      throw Exception("error in Sanitize CornerTexture!");
1025      return 0;
1026  }
1027
1028  int offset = 0;
1029  int glCornerTextureMipWidth = glCornerTextureWidth;
1030  int glCornerTextureMipHeight = glCornerTextureHeight;
1031
1032  for(int i = 0; i < num_levels; i++, offset += glCornerTextureMipWidth
*glCornerTextureMipHeight*4, glCornerTextureMipWidth/= 2,
glCornerTextureMipHeight/= 2)
1033  {
1034  glTexImage2D(GL_TEXTURE_2D, i, GL_RGBA, glCornerTextureMipWidth,
glCornerTextureMipHeight, 0, GL_RGBA, GL_FLOAT, &data2[offset]);
1035  }
1036  }
1037  }
1038  }
1039
1040  glutSetWindow(originalWin);
1041
1042  delete[] data1;
1043  delete[] data2;
1044
1045  return 1;
1046  }
1047
1048  int CreateTileTexture(const int numHColors,
1049  const int numVColors,
1050  const int numTilesPerColor,
1051  const int tileSize,
1052  GLuint & tileTextureID,
1053  GLuint & cornerTextureID
1054  )
1055  {
1056  glGenTextures(1, &tileTextureID);
1057  if(global.cornerSharpness > 0)
1058  {
1059      glGenTextures(1, & cornerTextureID);
1060  }
1061
1062  return CreateTileTexture(tileTextureID, cornerTextureID,
1063  numHColors, numVColors, numTilesPerColor,
1064  tileSize);
1065  }
1066

```

```

1067 void DumpMapping(const vector< vector<WangTiles::Tile> > & mapping)
1068 {
1069     const int height = mapping.size();
1070     const int width = mapping[0].size();
1071
1072     for(int i = height - 1; i >= 0; i--)
1073     {
1074     {
1075         for(int j = 0; j < width; j++)
1076         {
1077             cout << " " << mapping[i][j].e2() << " ";
1078         }
1079     }
1080
1081     cout << endl;
1082
1083     {
1084     for(int j = 0; j < width; j++)
1085     {
1086         cout << mapping[i][j].e3() << " " << mapping[i][j].e1() << " ";
1087     }
1088 }
1089
1090 cout << endl;
1091
1092 {
1093 for(int j = 0; j < width; j++)
1094 {
1095     cout << " " << mapping[i][j].e0() << " ";
1096 }
1097 }
1098
1099 cout << endl;
1100 }
1101 }
1102
1103 int CreateMapTexture(const GLuint tileMappingTextureID,
1104                     const GLuint cornerMappingTextureID,
1105                     const GLuint resultTextureID,
1106                     const int numHColors,
1107                     const int numVColors,
1108                     const int numTilesPerColor,
1109                     const int mappingTextureHeight,
1110                     const int mappingTextureWidth,
1111                     const int tileHeight,
1112                     const int tileWidth,
1113                     const int remapping)
1114 {
1115 // sequential tiling
1116 if(remapping ||
1117    (global.tileMapping.size() != mappingTextureHeight) ||
1118    (global.tileMapping[0].size() != mappingTextureWidth))
1119 {
1120     if(! WangTiles::SequentialTiling(*global.pTileSet,
1121                                     mappingTextureHeight,
1122                                     mappingTextureWidth,
1123                                     global.tileMapping) )
1124     {
1125         throw Exception("error in sequential tiling");
1126     }
1127 }
1128
1129 if(global.tileCompaction.size() <= 0)

```

```

1130 {
1131     return 0;
1132 }
1133
1134 const int tileTextureHeight = global.tileCompaction.size();
1135 const int tileTextureWidth = global.tileCompaction[0].size();
1136
1137 float *data = new float[mappingTextureHeight*mappingTextureWidth*4];
1138
1139 #if 0
1140     cout << "tile map" << endl;
1141     DumpMapping(global.tileMapping);
1142 #endif
1143 // loading LBM's Kinetics Energy.
1144 double *lbm = new double[mappingTextureHeight*mappingTextureWidth];
1145 ifstream lbm_file("KineticEnergy.txt");
1146
1147 if(!lbm_file){
1148     cout << endl << "Open LBM file Failure, It will be random value." <<
endl
;
1150     for(int i = 0;i< mappingTextureHeight * mappingTextureWidth;i++)
1151         lbm[i] = rand()*1.0/RAND_MAX;
1152     }else{
1153         double lbm_max, lbm_min;
1154         lbm_file >> lbm_max >> lbm_min;
1155         for(int j = 0;j< mappingTextureWidth;j++)
1156             for(int i = 0;i < mappingTextureHeight;i++)
1157                 {
1158                     int k = i*mappingTextureWidth + j;
1159                     lbm_file >> lbm[k];
1160                     lbm[k] = (lbm_max - lbm[k]) / (lbm_max - lbm_min);
1161                 }
1162
1163         lbm_file.close();
1164     }
1165
1166 // tile mapping texture
1167 {
1168     glActiveTextureARB(GL_TEXTURE0_ARB);
1169     glBindTexture(GL_TEXTURE_RECTANGLE_NV, tileMappingTextureID);
1170     glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
1171     glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
1172     for(int i = 0;i < mappingTextureHeight;i++)
1173         for(int j = 0;j < mappingTextureWidth;j++)
1174             {
1175                 int id = global.tileMapping[i][j].ID();
1176                 int row, col;
1177                 if(WangTiles::TileLocation(global.tileCompaction, id, row, col) != 1)
1178                     {
1179                         throw Exception("error in WangTiles::TileLocation()");
1180                     }
1181                 int op = i*mappingTextureWidth + j;
1182                 data[4*op + 0] = col;

```

```

1188 data[4*op + 1] = row;
1189 data[4*op + 2] = 0;
1190 data[4*op + 3] = float(lbm[op]); //0.9;
1191 }
1192
1193 glTexImage2D(GL_TEXTURE_RECTANGLE_NV, 0, GL_FLOAT_RGBA_NV,
mappingTextureWidth, mappingTextureHeight, 0, GL_RGBA, GL_FLOAT, data);
1194 }
1195
1196 // corner tiling texture
1197 if(cornerMappingTextureID > 0)
1198 {
1199 // create a corner compaction
1200 if(global.cornerCompaction.size() <= 0)
1201 {
1202 return 0;
1203 }
1204
1205 // corner tiling
1206 vector< vector<WangTiles::Tile> > cornerMapping;
1207
1208 if(! WangTiles::ShiftedCornerTiling(global.cornerCompaction,
1209 global.tileMapping,
1210 global.cornerMapping) )
1211 {
1212 throw Exception("error in shifted corner tiling");
1213 }
1214 #if 0
1215 //cout << "tile map" << endl;
1216 //DumpMapping(global.tileMapping);
1217
1218 cout << endl << "corner map" << endl;
1219 DumpMapping(global.cornerMapping);
1220 #endif
1221 glActiveTextureARB(GL_TEXTURE1_ARB);
1222 glBindTexture(GL_TEXTURE_RECTANGLE_NV, cornerMappingTextureID);
1223 glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
1224 glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
1225
1226 for(int i = 0; i < mappingTextureHeight; i++)
1227 for(int j = 0; j < mappingTextureWidth; j++)
1228 {
1229 int e0 = global.tileMapping[i][j].e1();
1230 int e1 = global.tileMapping[i][(j+1)%mappingTextureWidth].e2();
1231 int e2 = global.tileMapping[(i+1)%mappingTextureHeight][(j+1)%
mappingTextureWidth].e3();
1232 int e3 = global.tileMapping[(i+1)%mappingTextureHeight][j].e0();
1233
1234 int row, col;
1235
1236 if(WangTiles::CornerLocation(global.cornerCompaction,
1237 e0, e1, e2, e3, row, col) <= 0)
1238 {
1239 throw Exception("error in WangTiles::CornerLocation()");
1240 }
1241
1242 int op = i*mappingTextureWidth + j;
1243
1244 data[4*op + 0] = col;
1245 data[4*op + 1] = row;

```

```

1246 data[4*op + 2] = 0;
1247 data[4*op + 3] = 0;
1248 }
1249
1250 glTexImage2D(GL_TEXTURE_RECTANGLE_NV, 0, GL_FLOAT_RGBA_NV,
mappingTextureWidth, mappingTextureHeight, 1, GL_RGBA, GL_FLOAT, data);
1251 }
1252
1253 delete[] data;
1254 delete[] lbm;
1255
1256 if(resultTextureID > 0)
1257 {
1258     const int numVTiles = mappingTextureHeight;
1259     const int numHTiles = mappingTextureWidth;
1260
1261     const int resultTextureHeight = numVTiles * tileHeight;
1262     const int resultTextureWidth = numHTiles * tileWidth;
1263
1264     float *data = new float[resultTextureHeight*resultTextureWidth*4];
1265
1266     glActiveTextureARB(GL_TEXTURE0_ARB);
1267     glBindTexture(GL_TEXTURE_2D, resultTextureID);
1268     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
1269
1270     glTexParameterf(GL_TEXTURE_2D, GL_GENERATE_MIPMAP_SGIS, GL_TRUE);
1271     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR_MIPMAP_LINEAR);
1272     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
1273     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
1274     glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
1275
1276     for(int i = 0; i < numVTiles; i++)
1277     for(int j = 0; j < numHTiles; j++)
1278     {
1279         // build the tile
1280         vector<P4> colors(5);
1281
1282         int id = global.tileMapping[i][j].ID();
1283         int row, col;
1284
1285         if(WangTiles::TileLocation(global.tileCompaction, id, row, col) != 1)
1286         {
1287             throw Exception("error in WangTiles::TileLocation()");
1288         }
1289
1290         WangTiles::Tile compactionTile = global.tileCompaction[row][col];
1291
1292         vector< vector<P4> > > tile;
1293
1294         if(! global.tileCache.Get(compactionTile, tile))
1295         {
1296             throw Exception("Error in tileCache.Get()");
1297         }
1298
1299         // assign to the result texture
1300         for(int m = i*tileHeight; m < (i+1)*tileHeight; m++)
1301         for(int n = j*tileWidth; n < (j+1)*tileWidth; n++)
1302         {
1303             int op = m*resultTextureWidth + n;
1304             P4 color = tile[m - i*tileHeight][n - j*tileWidth];
1305

```

```

1306 data[4*op + 0] = color.x;
1307 data[4*op + 1] = color.y;
1308 data[4*op + 2] = color.z;
1309 data[4*op + 3] = color.w;
1310 }
1311 }
1312
1313 glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, resultTextureWidth,
resultTextureHeight, 0, GL_RGBA, GL_FLOAT, data);
1314
1315 delete[] data;
1316 }
1317
1318 return 1;
1319 }
1320
1321 int CreateMappingTexture(const int numHColors,
1322     const int numVColors,
1323     const int numTilesPerColor,
1324     const int mappingTextureHeight,
1325     const int mappingTextureWidth,
1326     const int tileHeight,
1327     const int tileWidth,
1328     GLuint & tileMappingTextureID,
1329     GLuint & cornerMappingTextureID,
1330     GLuint & resultTextureID)
1331 {
1332     glGenTextures(1, &tileMappingTextureID);
1333
1334     if(global.cornerSharpness > 0)
1335     {
1336         glGenTextures(1, &cornerMappingTextureID);
1337     }
1338
1339     if(global.permutationTextureSize < 0)
1340     {
1341         glGenTextures(1, &resultTextureID);
1342     }
1343
1344     return CreateMappingTexture(tileMappingTextureID,
1345         cornerMappingTextureID,
1346         resultTextureID,
1347         numHColors, numVColors, numTilesPerColor,
1348         mappingTextureHeight, mappingTextureWidth,
1349         tileHeight, tileWidth, 1);
1350 }
1351
1352 int CreatePermutationTexture(const GLuint permutationTextureID,
1353     const int permutationTextureSize)
1354 {
1355
1356     glBindTexture(GL_TEXTURE_RECTANGLE_NV, permutationTextureID);
1357     glTexParameteri(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_WRAP_S, GL_CLAMP);
1358     glTexParameteri(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_WRAP_T, GL_CLAMP);
1359     glTexParameteri(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
1360     glTexParameteri(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
1361
1362     float *permutationData = new float[permutationTextureSize*4];
1363
1364     vector<float> sequence(permutationTextureSize);
1365     {

```

```

1366 for(int i = 0;i < sequence.size();i++)
1367 {
1368     sequence[i] = i;
1369 }
1370 }
1371
1372 // random permutation
1373 for(int i = 0;i < permutationTextureSize;i++)
1374 {
1375     int selection = rand()%sequence.size();
1376
1377     permutationData[4*i] = permutationData[4*i + 1] = permutationData[4*i
+ 2] =
permutationData[4*i + 3] = sequence[selection];
1378
1379     sequence[selection] = sequence[sequence.size() - 1];
1380     sequence.pop_back();
1381 }
1382
1383 glTexImage2D(GL_TEXTURE_RECTANGLE_NV, 0, GL_FLOAT_RGBA_NV,
permutationTextureSize, 0, 0, GL_RGBA, GL_FLOAT, permutationData);
1384
1385 delete[] permutationData;
1386
1387 return permutationTextureID;
1388 }
1389
1390 int CreatePermutationTexture(const int permutationTextureSize)
1391 {
1392     GLuint handle;
1393     glGenTextures(1, &handle);
1394
1395     return CreatePermutationTexture(handle, permutationTextureSize);
1396 }
1397
1398 void Keyboard(unsigned char key, int x, int y)
1399 {
1400     switch(key)
1401     {
1402     case 'D':
1403     {
1404         int params[6];
1405         glGetIntegerv(GL_VIEWPORT, params);
1406         glutSetCursor(GLUT_CURSOR_WAIT);
1407         FrameBuffer::WriteColor(params[2], params[3], "snapshot.ppm");
1408         glutSetCursor(GLUT_CURSOR_INHERIT);
1409     }
1410     break;
1411
1412     case 27:
1413         throw ExitException("exit");
1414     break;
1415
1416     case 'r':
1417     case 'R':
1418     if(! TileCompaction(global.tileCompaction, key == 'r'))
1419     {
1420         // force random compaction
1421         throw Exception("wrong TileCompaction in Keyboard()");
1422     }
1423
1424     // follow through

```

```

1425
1426 default:
1427     CreateTileTexture(global.tilesTextureID,
1428         global.cornersTextureID,
1429         global.numHColors,
1430         global.numVColors,
1431         global.numTilesPerColor,
1432         global.tileSize);
1433
1434 if(glutGetWindow() == global.resultWindow)
1435 {
1436     if(pCgWangTilesT)
1437     {
1438         if(! CreateMappingTexture(global.tileMappingTextureID,
1439             global.cornerMappingTextureID,
1440             global.resultTextureID,
1441             global.numHColors,
1442             global.numVColors,
1443             global.numTilesPerColor,
1444             global.mappingTextureHeight,
1445             global.mappingTextureWidth,
1446             global.tileSize,
1447             global.tileSize,
1448             (key != 'r') && (key != 'R')))
1449         {
1450             throw Exception("error in CreateMappingTexture");
1451         }
1452     }
1453
1454     if(pCgWangTilesC)
1455     {
1456         if(! CreatePermutationTexture(global.permutationTextureID,
1457             global.permutationTextureSize))
1458         {
1459             throw Exception("error in CreatePermutationTexture");
1460         }
1461     }
1462 }
1463
1464 if(global.resultWindow > 0)
1465     glutPostWindowRedisplay(global.resultWindow);
1466 if(global.tileWindow > 0)
1467     glutPostWindowRedisplay(global.tileWindow);
1468 if(global.cornerWindow > 0)
1469     glutPostWindowRedisplay(global.cornerWindow);
1470
1471 break;
1472 }
1473 }
1474
1475 void Idle()
1476 {
1477     //if(glutGetWindow() == global.resultWindow)
1478     {
1479         if(global.timerWindow > 0)
1480             glutPostWindowRedisplay(global.timerWindow);
1481
1482         if(global.resultWindow > 0)
1483             glutPostWindowRedisplay(global.resultWindow);
1484     }
1485 }
1486
1487 int Main(int argc, char **argv)
1488 {

```

```
1489 int argCtr = 0;
1490 const int resultWinHeight = -1;
1491 const int resultWinWidth = -1;
1492 const int numVColors = 2;
1493 const int numHColors = 2;
1494 const int numTilesPerColor = 1;
1495 const char * tilePackingFileName = "cloud128.ppm";
1496 //const char * tilePackingFileName = "128";
1497 int tileSize = atoi(tilePackingFileName);
1498 const int mappingTextureHeight = 16;
1499 const int mappingTextureWidth = 16;
1500 const float cornerSharpness = 0.0;
1501 const int hashTableSize = 0;
1502 const int eventTableSize = -1;
1503 const int eventHistorySize = 0;
1504 const int compactionMethod = 1;
1505 const int viewOption = 0;
1506 const int sanitize = 0;
1507 const int randomSeed = -1;
1508
1509 srand(randomSeed >= 0 ? randomSeed : time(0));
1510
1511 {
1512 global.dumpAnimation = (viewOption == 3);
1513
1514 global.tilesTextureID = 0;
1515 global.tileMappingTextureID = 0;
1516 global.cornerMappingTextureID = 0;
1517 global.resultTextureID = 0;
1518
1519 global.numHColors = numHColors;
1520 global.numVColors = numVColors;
1521 global.numTilesPerColor = numTilesPerColor;
1522
1523 global.tileSize = tileSize;
1524 global.mappingTextureHeight = mappingTextureHeight;
1525 global.mappingTextureWidth = mappingTextureWidth;
1526 global.permutationTextureSize = hashTableSize;
1527
1528 global.reset = 0;
1529
1530 global.compaction = compactionMethod;
1531
1532 global.cornerSharpness = cornerSharpness;
1533
1534 global.eventHistory = eventHistorySize;
1535
1536 if(eventTableSize > 0)
1537 {
1538     global.pEventTimer = new EventTimer(eventTableSize);
1539 }
1540 else
1541 {
1542     global.pEventTimer = 0;
1543 }
1544
1545 if((viewOption == 2) || (viewOption == 3)) BuildVisualizationPath();
1546
1547 global.startTime = global.endTime = -1;
1548 global.numFrames = 0;
1549
```

```

1550 global.pTileSet = new WangTiles::TileSet(numHColors,
1551         numVColors,
1552         numTilesPerColor);
1553
1554 if(!TileCompaction(global.tileCompaction, global.compaction))
1555 {
1556     throw Exception("error in creating compaction");
1557 }
1558
1559 if(!CornerCompaction(global.cornerCompaction))
1560 {
1561     throw Exception("error in creating corner compaction");
1562 }
1563
1564 global.resultWindow = global.tileWindow = global.cornerWindow =
global.
timerWindow = 0;
1565
1566 if(tileSize <= 0)
1567 {
1568     int maximumValue = 0;
1569
1570     vector< vector<FrameBuffer::P3> > inputTilePacking;
1571
1572     if(!FrameBuffer::ReadPPM(tilePackingFileName,
1573         inputTilePacking, maximumValue))
1574     {
1575         throw Exception("error in reading tile packing");
1576     }
1577
1578     if(!CreateInputTileSet(inputTilePacking,
1579         maximumValue,
1580         numHColors,
1581         numVColors,
1582         numTilesPerColor,
1583         global.tileCache))
1584     {
1585         throw Exception("error in creating input tile set");
1586     }
1587
1588     if(global.tileCache.TileHeight() == global.tileCache.TileWidth())
1589     {
1590         tileSize = global.tileCache.TileHeight();
1591     }
1592     else
1593     {
1594         throw Exception("tileHeight != tileWidth");
1595     }
1596
1597     global.tileSize = tileSize;
1598     global.changeTileCache = 0;
1599 }
1600 else
1601 {
1602     global.changeTileCache = 1;
1603 }
1604
1605 // scene
1606 global.perspectiveView = (viewOption == 1);
1607
1608 global.winWidth = resultWinWidth > 0? resultWinWidth : tileSize *
mappingTextureWidth;
1609 global.winHeight = resultWinHeight > 0? resultWinHeight : tileSize *

```

```
mappingTextureHeight;
1610
1611   global.fov = 45;
1612
1613   global.eye[0] = 0;global.eye[1] = -1;global.eye[2] = 0.1;
1614
1615   global.center[0] = 0;global.center[1] = -0.7;global.center[2] = 0;
1616
1617   global.sanitize = sanitize;
1618   }
1619
1620   glutInit(&argc, argv);
1621   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
1622
1623   // tile-pack window
1624   glutInitWindowSize(tileSize * global.tileCompaction[0].size(),
1625   tileSize * global.tileCompaction.size());
1626   global.tileWindow = glutCreateWindow("Wang Tiles");
1627
1628   glutKeyboardFunc(Keyboard);
1629   glutDisplayFunc(Display);
1630
1631   // corner-pack window
1632   if(global.cornerSharpness > 0)
1633   {
1634   glutInitWindowSize(tileSize * global.cornerCompaction[0].size(),
1635   tileSize * global.cornerCompaction.size());
1636   global.cornerWindow = glutCreateWindow("Corner Tiles");
1637
1638   glutKeyboardFunc(Keyboard);
1639   glutDisplayFunc(Display);
1640   }
1641
1642   // timer window
1643   if(global.pEventTimer)
1644   {
1645   glutInitWindowSize(128, 128);
1646   global.timerWindow = glutCreateWindow("Frame Rate");
1647
1648   glutKeyboardFunc(Keyboard);
1649   glutDisplayFunc(TimerDisplay);
1650   }
1651
1652   // result tiling window
1653   glutInitWindowSize(global.winWidth, global.winHeight);
1654   global.resultWindow = glutCreateWindow("Tiling");
1655
1656   glutKeyboardFunc(Keyboard);
1657   glutDisplayFunc(Display);
1658   if(global.timerWindow || (global.cameraPath.size() > 0))
1659   {
1660   glutIdleFunc(Idle);
1661   }
1662
1663   if(!glh_init_extensions(REQUIRED_EXTENSIONS))
1664   {
1665   throw Exception("Necessary extensions were not supported");
1666   }
1667
1668   glDisable(GL_LIGHTING);
1669   glEnable(GL_DEPTH_TEST);
1670
```

```

1671  clock_t t1 = clock();
1672  if(! CreateTileTexture(numHColors, numVColors, numTilesPerColor,
1673  global.tileSize,
1674  global.tilesTextureID, global.cornersTextureID))
1675  {
1676  throw Exception("error in creating tile textures");
1677  }
1678  clock_t t2 = clock();
1679  cout << "Time CreateTileText: " << double(t2-
1680  t1)/(double)CLOCKS_PER_SEC << endl;
1681  if(hashTableSize > 0)
1682  {
1683  global.permutationTextureID = \
1684  CreatePermutationTexture(hashTableSize);
1685
1686  pCgWangTilesC = new CgWangTilesC(global.tilesTextureID,
1687  global.cornersTextureID,
1688  tileSize, tileSize,
1689  numHColors*numHColors,
1690  numVColors*numVColors,
1691  numVColors*numVColors,
1692  numHColors*numHColors,
1693  mappingTextureHeight,
1694  mappingTextureWidth,
1695  global.permutationTextureID,
1696  cornerSharpness);
1697  }
1698  else
1699  {
1700  clock_t t3=clock();
1701  if(! CreateMappingTexture(numHColors, numVColors, numTilesPerColor,
1702  mappingTextureHeight,
1703  mappingTextureWidth,
1704  tileSize,
1705  tileSize,
1706  global.tileMappingTextureID,
1707  global.cornerMappingTextureID,
1708  global.resultTextureID))
1709  {
1710  throw Exception("error in creating mapping textures");
1711  }
1712
1713  if(hashTableSize == 0)
1714  {
1715  pCgWangTilesT = new CgWangTilesT(global.tilesTextureID,
1716  global.cornersTextureID,
1717  tileSize, tileSize,
1718  global.tileMappingTextureID,
1719  global.cornerMappingTextureID,
1720  cornerSharpness);
1721  }
1722  else
1723  {
1724  pCgWangTilesT = 0;
1725  }
1726  clock_t t4=clock();
1727  cout << "Time CreateMappingTexture: " << double(t4-
1728  t3)/(double)CLOCKS_PER_SEC << endl;
1729  }
1730  Enable();
1731

```

```

1732  glutMainLoop();
1733
1734  if(pCgWangTilesT) delete pCgWangTilesT;
1735  if(pCgWangTilesC) delete pCgWangTilesC;
1736
1737  return 0;
1738 }
1739
1740 int main(int argc, char **argv)
1741 {
1742     try
1743     {
1744         return Main(argc, argv);
1745     }
1746     catch(ExitException e)
1747     {
1748         if(global.pEventTimer)
1749         {
1750             // currently, the frame rate only makes sense if you turn on
event timer.
1751             global.endTime = global.timer.CurrentTime();
1752
1753             cout << "frame rate is " << global.numFrames/(global.endTime -
global.startTime) << endl;
1754         }
1755
1756         if(pCgWangTilesT) delete pCgWangTilesT;
1757         if(pCgWangTilesC) delete pCgWangTilesC;
1758
1759         return 0;
1760     }
1761     catch(Exception e)
1762     {
1763         cerr<<"Error : "<<e.Message()<<endl;
1764         return 1;
1765     }
1766 }
1767

```

การปรับปรุงโปรแกรม CgWangTileT.cpp

```

1 /*
2 * CgWangTilesT.cpp
3 *
4 * Li-Yi Wei
5 * 8/9/2003
6 *
7 * Modify by Mr. Supachai 2010 *
8 */
9
10 #include "CgWangTilesT.hpp"
11
12 #include <iostream>
13 #include <sstream>
14
15 using namespace std;
16
17 #include <glh/glh_extensions.h>
18
19 #define REQUIRED_EXTENSIONS "WGL_ARB_pbuffer " \

```

```

20         "WGL_ARB_pixel_format " \
21         "WGL_ARB_render_texture " \
22         "GL_NV_float_buffer " \
23         "GL_NV_texture_rectangle " \
24         "GL_ARB_multitexture "
25
26 CgWangTilesT::CgWangTilesT(const GLuint tilesTextureID,
27         const GLuint cornersTextureID,
28         const int tileHeight,
29         const int tileWidth,
30         const GLuint tileMappingTextureID,
31         const GLuint cornerMappingTextureID,
32         const float cornerSharpness) throw(Exception) : _context(0)
, _program(0), _tilesTexture(0), _cornersTexture(0), _tileMappingTexture(0),
 _cornerMappingTexture(0)
33 {
34     if( (tilesTextureID <= 0) && (tileMappingTextureID > 0)) ||
35     ((tilesTextureID > 0) && (tileMappingTextureID <= 0)) ||
36     ((cornersTextureID <= 0) && (cornerMappingTextureID > 0)) ||
37     ((cornersTextureID > 0) && (cornerMappingTextureID <= 0)) )
38     {
39         throw Exception("CgWangTilesT: illegal tile or corner selection!");
40     }
41
42     cgSetErrorCallback(CheckError);
43
44     _context = cgCreateContext();
45
46     if(!_context)
47     {
48         throw Exception("null context");
49     }
50
51     _fragmentProfile = cgGLGetLatestProfile(CG_GL_FRAGMENT);
52     cgGLSetOptimalOptions(_fragmentProfile);
53
54     // tile texture size
55     int tileTextureHeight = 0;
56     int tileTextureWidth = 0;
57
58     if(tilesTextureID > 0)
59     {
60         glActiveTextureARB(GL_TEXTURE0_ARB);
61         glBindTexture(GL_TEXTURE_2D, tilesTextureID);
62
63         glGetTexLevelParameteriv(GL_TEXTURE_2D, 0,
64         GL_TEXTURE_HEIGHT, &tileTextureHeight);
65         glGetTexLevelParameteriv(GL_TEXTURE_2D, 0,
66         GL_TEXTURE_WIDTH, &tileTextureWidth);
67
68         if( (tileTextureHeight <= 0) || (tileTextureWidth <= 0) )
69         {
70             throw Exception("CgWangTilesT: illegal tile texture size");
71         }
72
73         tileTextureHeight /= tileHeight;
74         tileTextureWidth /= tileWidth;
75     }
76
77     // corner texture size
78     int cornerTextureHeight = 0;
79     int cornerTextureWidth = 0;
80

```

```

81     if(cornersTextureID > 0)
82     {
83         glActiveTextureARB(GL_TEXTURE1_ARB);
84         glBindTexture(GL_TEXTURE_2D, cornersTextureID);
85
86         glGetTexLevelParameteriv(GL_TEXTURE_2D, 0,
87             GL_TEXTURE_HEIGHT, &cornerTextureHeight);
88         glGetTexLevelParameteriv(GL_TEXTURE_2D, 0,
89             GL_TEXTURE_WIDTH, &cornerTextureWidth);
90
91         if( (cornerTextureHeight <= 0) || (cornerTextureWidth <= 0) )
92         {
93             throw Exception("CgWangTilesT: illegal corner texture size");
94         }
95
96         cornerTextureHeight /= tileHeight;
97         cornerTextureWidth /= tileWidth;
98     }
99
100    // tile mapping texture size
101    int tileMappingTextureHeight = 0;
102    int tileMappingTextureWidth = 0;
103
104    if(tileMappingTextureID > 0)
105    {
106        glActiveTextureARB(GL_TEXTURE0_ARB);
107        glBindTexture(GL_TEXTURE_RECTANGLE_NV, tileMappingTextureID);
108
109        glGetTexLevelParameteriv(GL_TEXTURE_RECTANGLE_NV, 0,
110            GL_TEXTURE_HEIGHT, &tileMappingTextureHeight);
111        glGetTexLevelParameteriv(GL_TEXTURE_RECTANGLE_NV, 0,
112            GL_TEXTURE_WIDTH, &tileMappingTextureWidth);
113
114        if( (tileMappingTextureHeight <= 0) || (tileMappingTextureWidth
115    <= 0) )
116        {
117            throw Exception("CgWangTilesT: illegal tile mapping texture
118    size");
119        }
120    }
121
122    // corner mapping texture size
123    int cornerMappingTextureHeight = 0;
124    int cornerMappingTextureWidth = 0;
125
126    if(cornerMappingTextureID > 0)
127    {
128        glActiveTextureARB(GL_TEXTURE1_ARB);
129        glBindTexture(GL_TEXTURE_RECTANGLE_NV, cornerMappingTextureID);
130
131        glGetTexLevelParameteriv(GL_TEXTURE_RECTANGLE_NV, 0,
132            GL_TEXTURE_HEIGHT, &cornerMappingTextureHeight);
133        glGetTexLevelParameteriv(GL_TEXTURE_RECTANGLE_NV, 0,
134            GL_TEXTURE_WIDTH, &cornerMappingTextureWidth);
135
136        if( (cornerMappingTextureHeight <= 0) ||
137    (cornerMappingTextureWidth <= 0) )
138        {
139            throw Exception("CgWangTilesT: illegal corner mapping
140    texture size");
141        }
142    }
143
144    }
145
146    }

```

```

140     if((tileMappingTextureHeight &
141         ornerMappingTextureHeight &
142         (tileMappingTextureHeight != cornerMappingTextureHeight)) ||
143         (tileMappingTextureWidth &
144         cornerMappingTextureWidth &
145         (tileMappingTextureWidth != cornerMappingTextureWidth))
146     )
147     {
148         throw Exception("CgWangTilesT: incompatible tile and corner
mapping texturesizes");
149     }
150
151     string program = \
152         CreateProgram(tileHeight, tileWidth,
153             tileTextureHeight, tileTextureWidth,
154             cornerTextureHeight, cornerTextureWidth,
155             tileMappingTextureHeight, tileMappingTextureWidth,
156             cornerSharpness);
157
158     _program = cgCreateProgram(_context, CG_SOURCE, program.c_str(),
_fragmentProfile,"fragment", 0);
159
160     #if 0
161         cout << program << endl;
162         cout<<"---- PROGRAM BEGIN ----"<<endl;
163         cout<<cgGetProgramString(_program, CG_COMPILED_PROGRAM);
164         cout<<"---- PROGRAM END ----"<<endl;
165     #endif
166
167     if(_program)
168     {
169         cgGLLoadProgram(_program);
170
171         CheckError();
172     }
173
174     // tile texture
175     if(tilesTextureID > 0)
176     {
177         glActiveTextureARB(GL_TEXTURE0_ARB);
178         _tilesTexture = cgGetNamedParameter(_program, "tilesTexture");
179         cgGLSetTextureParameter(_tilesTexture, tilesTextureID);
180         cgGLEnableTextureParameter(_tilesTexture);
181     }
182
183     // corner texture
184     if(cornersTextureID > 0)
185     {
186         glActiveTextureARB(GL_TEXTURE1_ARB);
187         _cornersTexture = cgGetNamedParameter(_program, "cornersTexture");
188         cgGLSetTextureParameter(_cornersTexture, cornersTextureID);
189         cgGLEnableTextureParameter(_cornersTexture);
190     }
191
192     // tile mapping texture
193     if(tileMappingTextureID > 0)
194     {
195         _tileMappingTexture = cgGetNamedParameter(_program,
"tileMappingTexture");
196         cgGLSetTextureParameter(_tileMappingTexture, tileMappingTextureID);
197
198         cgGLEnableTextureParameter(_tileMappingTexture);
199     }

```

```

200
201 // corner mapping texture
202 if(cornerMappingTextureID > 0)
203 {
204     _cornerMappingTexture = cgGetNamedParameter(_program,
205         "cornerMappingTexture");
206     cgGLSetTextureParameter(_cornerMappingTexture,
207         cornerMappingTextureID);
208     cgGLEnableTextureParameter(_cornerMappingTexture);
209 }
210 // enable stuff
211 cgGLEnableProfile(_fragmentProfile);
212
213 cgGLBindProgram(_program);
214 }
215
216 CgWangTilesT::~CgWangTilesT(void)
217 {
218     Disable();
219     cgDestroyContext(_context);
220 }
221
222 void CgWangTilesT::CheckError(void)
223 {
224     CGError error = cgGetError();
225     if(error != CG_NO_ERROR)
226     {
227         throw Exception(cgGetErrorString(error));
228     }
229 }
230 }
231
232 void CgWangTilesT::Enable(void) const
233 {
234     if(_tileMappingTexture)
235         cgGLEnableTextureParameter(_tileMappingTexture);
236     if(_cornerMappingTexture)
237         cgGLEnableTextureParameter(_cornerMappingTexture);
238     if(_tilesTexture) cgGLEnableTextureParameter(_tilesTexture);
239     if(_cornersTexture) cgGLEnableTextureParameter(_cornersTexture);
240     cgGLEnableProfile(_fragmentProfile);
241     cgGLBindProgram(_program);
242 }
243
244 void CgWangTilesT::Disable(void) const
245 {
246     if(_tileMappingTexture)
247         cgGLDisableTextureParameter(_tileMappingTexture);
248     if(_cornerMappingTexture)
249         cgGLDisableTextureParameter(_cornerMappingTexture);
250     if(_tilesTexture) cgGLDisableTextureParameter(_tilesTexture);
251     if(_cornersTexture) cgGLDisableTextureParameter(_cornersTexture);
252     cgGLDisableProfile(_fragmentProfile);
253 }
254
255 string CgWangTilesT::CreateProgram(const int tileHeight,
256     const int tileWidth,
257     const int tileTextureHeight,

```

```

257         const int tileTextureWidth,
258         const int cornerTextureHeight,
259         const int cornerTextureWidth,
260         const int tileMappingTextureHeight,
261         const int tileMappingTextureWidth,
262         const float cornerSharpness)
263 {
264     stringstream strResult;
265
266     strResult << "struct FragmentInput" << endl;
267     strResult << "{" << endl;
268     strResult << " float4 tex : TEX0;" << endl;
269     strResult << " float4 col : COL0;" << endl;
270     strResult << "};" << endl;
271     strResult << "struct FragmentOutput" << endl;
272     strResult << "{" << endl;
273     strResult << " float4 col : COL;" << endl;
274     strResult << "};" << endl;
275     strResult << "" << endl;
276     strResult << "float2 mod(const float2 a, const float2 b)" << endl;
277     strResult << "{" << endl;
278     strResult << " return floor(frac(a/b)*b);" << endl;
279     strResult << "}" << endl;
280     strResult << "" << endl;
281     strResult << "FragmentOutput fragment(FragmentInput input," << endl;
282     strResult << " uniform sampler2D tilesTexture," << endl;
283     strResult << " uniform sampler2D cornersTexture," << endl;
284     strResult << " uniform samplerRECT tileMappingTexture," << endl;
285     strResult << " uniform samplerRECT cornerMappingTexture)" << endl;
286     strResult << "{" << endl;
287     strResult << " FragmentOutput output;" << endl;
288
289     strResult << " float2 mappingScale = float2(" <<
tileMappingTextureWidth << ",
" << tileMappingTextureHeight << ");" << endl;
290     strResult << " float2 mappingAddress = input.tex.xy * mappingScale;"
<<endl;
291
292     if( (tileTextureHeight > 0) && (tileTextureWidth > 0))
293     {
294         strResult << " float2 tileScale = float2(" << tileTextureWidth
<< ", " << tileTextureHeight << ");" << endl;
295         strResult << " float2 tileScaledTex = input.tex.xy * float2("
<< tileMappingTextureWidth*1.0/tileTextureWidth << ", " <<
tileMappingTextureHeight*1.0/tileTextureHeight << ");" << endl;
296         strResult << " float4 whichTile = texRECT(tileMappingTexture, mod
(mappingAddress, mappingScale));" << endl;
297         strResult << " float4 result1 = tex2D(tilesTexture, (whichTile.xy +
frac(mappingAddress))/tileScale, ddx(tileScaledTex), ddy(tileScaledTex));" <<
endl;
298
299         strResult << " output.col =
lerp(result1,float4(0.207,0.368,0.549,1.0),whichTile.a);" << endl;
300     }
301
302     if( (cornerTextureHeight > 0) && (cornerTextureWidth > 0))
303     {
304         strResult << " float2 cornerScale = float2(" << cornerTextureWidth
<< ", " << cornerTextureHeight << ");" << endl;
305         strResult << " float2 cornerScaledTex = input.tex.xy * float2(" <<

```

```

        tileMappingTextureWidth*1.0/cornerTextureWidth << ", " <<
        tileMappingTextureHeight*1.0/cornerTextureHeight << ");" << endl;
306     strResult << " mappingAddress.xy = mappingAddress.xy - float2(0.5,
0.5);" << endl;
307     strResult << " float4 whichCornerTile = texRECT(cornerMappingTexture,
mod(mappingAddress, mappingScale));" << endl;
308     strResult << " float4 result2 = tex2D(cornersTexture,
(whichCornerTile.xy + frac(mappingAddress) + float2(0.5, 0.5))/cornerScale,
ddx(cornerScaledTex), ddy(cornerScaledTex));" << endl;
309     strResult << " output.col = result2;" << endl;
310     }
311
312     if( (tileTextureHeight > 0) && (tileTextureWidth > 0) &&
313     (cornerTextureHeight > 0) && (cornerTextureWidth > 0) )
314     {
315     strResult << " float2 cornerDistance = frac(mappingAddress) -
float2(0.5, 0.5);" << endl;
316     strResult << " float bweight = exp(-dot(cornerDistance,
cornerDistance)/" << cornerSharpness << ");" << endl;
317     strResult << " output.col = result1*(1 - bweight) + result2*bweight;"
<<endl;
318
319     }
320
321     strResult << " return output;" << endl;
322     strResult << "}" << endl;
323
324     // done
325     string result(strResult.str(), strResult.pcount());
326     strResult.rdbuf()->freeze(0);
327     return result;
328 }
329

```