EXHIBIT 20

```
1  /*
2   * Name: *  TsTileStruct.c * * Description: Almost all of the tile table management
       is conducted in
3   *      this file. The tiles are read in and stored within the file. The
4   *      state of the tiles is also handled within this file.
5   *
6   * Function List:
7   *      TsCreateTileStruct                  externally available
8   *      TsTableStatus                       static
9   *  TsCloseTileStruct       externally available
10  *      TsGetTileData                       externally available
11  *  TsGetTile           externally available
12  *  TsGetTileStatus         externally available
13  *      TsCheckBoundary                     externally available
14  *      TsImageServerReader                 externally available
15  *      TsReadDems                          externally available
16  *      TsDrawTileMgr                       externally available
17  *      TsDrawTileMgrAge                    externally available
18  *      TsFillTileData                      externally available
19  *      TsClearData                         externally available
20  *      TsGetTileMgr                        externally available
21  *      TsConvertOiTiles                    externally available
22  *      TsGetNormal                         externally available
23  *      TsGetDemVal                         externally available
24  *      TsGetDEMNormal                      externally available
25  *  TsCleanUp           static
26  *      TsCleanTileTable                    static
27  *      TsCreateTileTable                   static
28  *      TsReadDEMLevel                      static
29  *      TsResetDataSet                      externally available
30  *      TsInitializeTileStruct              externally available
31  *      TsGetTspecInfo                      externally available
32  *      TsCreateTileSetInfo                 static
33  *      TsCreateDataSetInfo                 static
34  *      TsGetTileSetInfo                    externally available
35  *      TsGetDataSetInfo                    externally available
36  *      TsWriteReaderLogFile                static
37  *      TsReadTileFromTSM                   static
38  *
39  *
40  * Dependencies:
41  *  None
42  *
43  * Revision History:
44  *  $Date: 1996/04/27 11:34:10 $
45  *  $Author: lau $
46  */
47 #include "TsLogging.h"
48 #include "TsPrivate.h"
49
50 static char TsTileStruct_c_rcsid[] = "$Header: /homedir/magic/software/CVS/
      TerraVision/libTileSvc/TsTileStruct.c,v 2.43 1996/04/27 11:34:10 lau Exp $";
51 static char TsTileStruct_c_version[] = "$Revision: 2.43 $";
52 static char TsTileStruct_c_date[] = "$Date: 1996/04/27 11:34:10 $";
53
54 /* GLOBALLY VISIBLE VARIABLES */
55 volatile pid_t   procPid[MAX_PROCS];    /* PROCESS ID'S OF CHILDREN PROCESSES */
56 volatile int     numProcs = 0;          /* NUMBER OF CHILDREN PROCESSES       */
57 int      power2[NUM_POWER2];            /* PRE-COMPUTED POWERS OF 2           */
58
59 /* LOCAL STATIC VARIABLES */
60 static TsTileStruct *   tileStruct;     /* TILE TABLE INFORMATION             */
61
62 /* LOCAL FUNCTION PROTOTYPES */
63 static void TsCleanUp(usptr_t *);
64 static void TsCleanTileTable(TsTileType, usptr_t *);
65 static int  TsCreateTileTable(TsSetInfo *, TsTileType, usptr_t *);
```

```
 * Synopsis:
 *    void
 *    TsImageServerReader(void * data)
 *
 * Description: This function is spawned off as a seperate thread which
 *    reads in the tiles from the TSM and puts them in the tile tables.
 *    An instance of this function is spawned for each TSM server that
 *    is available.
 *
 * Externals:
 *            tileStruct - Pointer to the tile tables and associated
 *                         data structures
 *            logFlag - flag for logging timing information
 *            flushFlag - flag for flushing the timing information
 *                        periodically.
 *
 * Returns: NONE - This function should NEVER return.
 *
 * Author:
 *      Stephen Lau
 *
 * Date:
 *    July 8, 1993
 *
 */
void
TsImageServerReader(void * threadData)
{
 static char * funcName = "TsImageServerReader()";
 FILE *         logFile;
 ReaderStatus   readerStat;
 int            serverNum;
 char           logBuffer[100];
 extern int     logFlag;

    /* GET THE SERVER NUMBER */
    serverNum = (int) threadData;

    /* OPEN THE LOG FILE IF REQUESTED */
    if(logFlag)
    {
        sprintf(logBuffer,"/usr/tmp/TerraVisionReader%d.log", serverNum);
    logFile = fopen(logBuffer,"ab");
    if(!logFile)
    {
        my_status("%s: Unable "
              "to open log file %s. Not logging this run.",
              funcName,
              logBuffer);
        }
    }

    /* INITIALIZE THE READER STAT KEEPER */
    readerStat.tileBurstTime = 0;
    readerStat.timePerTile = 0;
    readerStat.totalBytes50 = 0;
    readerStat.totalBytesRead = 0;
    readerStat.tilesRead = 0;

    /* LOOP FOREVER */
    while(1)
    {
        if(!(readerStat.tilesRead % 50))
    {
        readerStat.totalBytes50 = 0;
            gettimeofday(&readerStat.time50[0]);
    }
```

```
668
669         /* READ TILE FROM THE TSM */
670     TsReadTileFromTSM(serverNum, &readerStat, logFile);
671     }
672 }
673
674 /*
675  * Synopsis:  static void TsReadTileFromTSM(int, ReaderStatus *, FILE *)
676  *
677  *
678  * Description: Reads a tile from the TSM and puts it in the tile
679  *    tables. If we want logging information, the logging information
680  *    is written to disk.
681  *
682  * Externals: TsTileStruct * tileStruct
683  *
684  *
685  * Returns: NONE
686  *
687  *
688  * Author:
689  *      Stephen Lau
690  *
691  * Date: December 12, 1994
692  *
693  */
694 static void
695 TsReadTileFromTSM(int            serverNum,
696           ReaderStatus * readerStat,
697           FILE *         logFile)
698 {
699  static char * funcName = "TsReadTileFromTSM()";
700  TileHeader           tileHeader;
701  TsTileSetInfo *      tileSetInfo;
702  extern RequestRecord requestRecord;
703  TileMgr *        tileMgr;
704  int                  errRet;
705  TileData             tileRequest;
706  struct timeval currTime;
707  extern TSMConnection * tsm;
708  extern TVLogger * tvLogger;
709
710     gettimeofday(&readerStat->burstTime[0]);
711
712 #ifdef DEBUG
713     my_status("%s: About to read a tile header.",
714           funcName);
715 #endif
716     /* READ THE HEADER */
717     errRet = tsmGetNextTileHeader(tsm->tsmh, &tileHeader, serverNum);
718     if(!errRet)
719     {
720 #ifdef DEBUG
721     my_status("%s: Didn't read header %d",
722           funcName, errRet);
723 #endif
724     return;
725     }
726
727     /*        *****  THIS IS REALLY UGLY  *****
728     /* THE ISS STRUCT "TILEHEADER" DIVERGED FROM WHAT WE HAD ORIGINALLY
729      * SPECFIED AND IS A FIXED SIZE. THE "TYPE" FIELD WAS DROPPED
730      * WHICH CAUSED TV TO LOSE INFORMATION. THIS ATTEMPTS TO BRIDGE
731      * THE GAP.
732      */
733     tileRequest.id.x = tileHeader.ar.x;
734     tileRequest.id.y = tileHeader.ar.y;
```

```
735     tileRequest.id.res = tileHeader.ar.r;
736     tileRequest.id.setId = tileHeader.ar.set_id;
737     tileRequest.id.sid = tileHeader.ar.session_id;
738     tileRequest.id.tile_id = tileHeader.ar.tile_id;
739 #ifdef DEBUG
740     my_status("%s: Reading tile at %d %d %d %d %d %d.",
741           funcName,
742           tileRequest.id.x, tileRequest.id.y,
743           tileRequest.id.res, tileRequest.id.setId,
744           tileRequest.id.sid, tileRequest.id.tile_id);
745 #endif
746     my_status("%s: Reading tsm tile at %d %d %d %d %d %d.",
747           funcName,
748           tileHeader.ar.x, tileHeader.ar.y,
749           tileHeader.ar.r, tileHeader.ar.set_id,
750           tileHeader.ar.session_id, tileHeader.ar.tile_id);
751
752     tileRequest.id.type = TSMGetTypeFromSetId(tsm, tileRequest.id.setId);
753     if(tileRequest.id.type == -1)
754     {
755     my_status("%s: Unknown tile type from bogus setId %d.",
756           funcName,
757           tileRequest.id.setId);
758     exit(1);
759     }
760 #ifdef DEBUG
761     my_status("%s: Reading tile at %d %d %d %d.",
762           funcName,
763           tileRequest.id.x, tileRequest.id.y,
764           tileRequest.id.res, tileRequest.id.setId);
765 #endif
766
767     /* GET THE TILE SET INFO FOR THE TILE */
768     tileSetInfo = TsGetTileSetInfo(tileRequest.id.type);
769     if(!tileSetInfo)
770     {
771         my_status("%s: Invalid data set specified!",
772           funcName);
773     return;
774     }
775
776     /* ALLOCATE SPACE FOR THE INCOMING DATA */
777     tileRequest.data = (char *) usmalloc(tileSetInfo->tileSize,
778                       tileStruct->arena);
779     if(!tileRequest.data)
780     {
781         my_status("%s: Unable to allocate "
782           "space for the tile data!",
783           funcName);
784     exit(1);
785     }
786 #ifdef DEBUG
787     my_status("%s: About to read Tile", funcName);
788 #endif
789     if(!tsmGetNextTileData(tsm->tsmh, &tileHeader, tileRequest.data))
790     {
791         my_status("%s: Unable to read the tile data! %d",
792           funcName, errRet);
793     exit(1);
794     }
795 #ifdef DEBUG
796     my_status("%s: Rad Tile", funcName);
797 #endif
798
799     /* CALCULATE THE BURST RATE */
800     gettimeofday(&readerStat->burstTime[1]);
801     readerStat->tileBurstTime = t_elapsed_time(&readerStat->burstTime[0],
```

```
802                             &readerStat->burstTime[1]);
803
804     /* DO SOME READER TSM_BOOKKEEPING */
805     if(readerStat->tileBurstTime < tsm->issInfo.maxBursts[serverNum])
806     {
807         tsm->issInfo.maxBursts[serverNum] = readerStat->tileBurstTime;
808     }
809     readerStat->timePerTile += readerStat->tileBurstTime;
810     requestRecord.amtReceived += tileSetInfo->tileSize + sizeof(TileHeader);
811     readerStat->totalBytes50 += tileSetInfo->tileSize + sizeof(TileHeader);
812     readerStat->totalBytesRead += tileSetInfo->tileSize + sizeof(TileHeader);
813     readerStat->tilesRead++;
814
815     /* SPIT OUT THE LOG IF WE WANT LOGGING */
816     if(logFile)
817     {
818         TsWriteReaderLogFile(&tileRequest,
819                 serverNum,
820                 readerStat,
821                 logFile,
822                 tsm);
823     }
824
825     if(tvLogger->logging[TV_READER_LOG])
826     {
827         gettimeofday(&currTime);
828 #ifdef DEBUG
829     my_status("%s: Logging %d", funcName, serverNum);
830 #endif
831     TVLoggerTileRead(tvLogger, serverNum, &currTime);
832     }
833
834     /* KEEP TRACK OF TSM PERFORMANCE */
835     if(!(readerStat->tilesRead % 50) && readerStat->tilesRead >= 50)
836     {
837         gettimeofday(&readerStat->time50[1]);
838     tsm->issInfo.appAverage[serverNum] =
839        (float) (8.0 * readerStat->totalBytesRead) /
840           (float) t_elapsed_time(&readerStat->time50[0],
841                      &readerStat->time50[1]) / 1000000;
842     tsm->issInfo.netAverage[serverNum] =
843        (float) (8.0 * readerStat->totalBytes50) /
844           readerStat->timePerTile / 1000000;
845     tsm->issInfo.maxBursts[serverNum] =
846        readerStat->timePerTile / 50;
847 #ifdef DEBUG
848     my_status("%s: Net = %4.4f App = %4.4f",
849           funcName,
850           tsm->issInfo.netAverage[serverNum],
851           tsm->issInfo.appAverage[serverNum]);
852 #endif
853     readerStat->timePerTile = 0;
854     readerStat->totalBytes50 = 0;
855     }
856
857     /* MAKE SURE THE TILE DATA IS NOT CORRUPTED */
858 #ifdef DEBUG
859     if (iss_checksum((u_short *) tileRequest.data, 1024) !=
860     tileHeader.check_sum)
861     {
862         my_status("%s: Error in checksum for "
863           "tile at %d %d %d.",
864           funcName,
865           tileRequest.id.x, tileRequest.id.y,
866           tileRequest.id.res);
867 #ifdef DEBUG
868     return;
```

```
#endif
    }
#endif

    /* FIND THE TILE MANAGER AND PUT THE TILE INTO IT */
    tileMgr = tileStruct->tileMgrs[tileRequest.id.type];
    if(!TileMgrCheckAvailability(tileMgr,
                &tileRequest.id,
                &tileRequest,
                TimerSeconds()))
    {
        my_status("%s: Unable to allocate "
          "space for tile in tile manager!",
          funcName);
    exit(1);
    }
#ifdef DEBUG
    for(i = 0, j = 0; i < 128 * 128; i++)
    {
    b = tileRequest.data[j++] & 0xFF;
    g = tileRequest.data[j++] & 0xFF;
    r = tileRequest.data[j++] & 0xFF;
    imageBuffer[serverNum][i] = (unsigned long)
      ((b << 16) & 0x00FF0000) +
        ((g << 8)  & 0x0000FF00) +
          (r & 0x0000FF);
    }
#endif
}

/*
 * Syncpsis: void TsWriteReaderLogFile(TileData *,
 *                                     int, ReaderStatus*, FILE *)
 *
 *
 * Description: Write logging and timing information to a file that is
 *    specified by FILE *
 *
 * Externals: flushFlag - flag wether or not we need to flush the stream
 *
 *
 * Returns: NONE
 *
 * Author:
 *      Stephen Lau
 *
 * Date: December 12, 1994
 *
 */
static void
TsWriteReaderLogFile(TileData *     tileRequest,
             int            proc,
             ReaderStatus * readerStat,
             FILE *         logFile,
             volatile TSMConnection * tsm)
{
 static char * funcName = "TsWriteReaderLogFile()";
 struct timeval currTime;
 char           logBuffer[1000];
 extern int     flushFlag;

    gettimeofday(&currTime);
    sprintf(logBuffer, "Receive Time = %f\tRequest Time ="
        " %f\tLatency = %f\t",

        (double) currTime.tv_sec + ((double) currTime.tv_usec
                    / 1000000.0),
```

```
936             (double) tileRequest->tv[SENT].tv_sec +
937             ((double) tileRequest->tv[SENT].tv_usec / 1000000.0),
938             t_elapsed_time(&tileRequest->tv[SENT],&currTime));
939         fwrite(logBuffer, sizeof(char), strlen(logBuffer), logFile);
940         sprintf(logBuffer,"Master Out = %f\tMaster In = "
941             "%f\tServer In = %f\tStart Read = %f\t"
942             "End Read = %f\tStart Write = %f\tEnd Write = "
943             "%f\tTSM Received = %f\n",
944             (double) tileRequest->tv[MASTER_OUT].tv_sec +
945             ((double) tileRequest->tv[MASTER_OUT].tv_usec / 1000000.0),
946             (double) tileRequest->tv[MASTER_IN].tv_sec +
947             ((double) tileRequest->tv[MASTER_IN].tv_usec / 1000000.0),
948             (double) tileRequest->tv[SERVIN].tv_sec +
949             ((double) tileRequest->tv[SERVIN].tv_usec / 1000000.0),
950             (double) tileRequest->tv[STARTREAD].tv_sec +
951             ((double) tileRequest->tv[STARTREAD].tv_usec / 1000000.0),
952             (double) tileRequest->tv[ENDREAD].tv_sec +
953             ((double) tileRequest->tv[ENDREAD].tv_usec / 1000000.0),
954             (double) tileRequest->tv[STARTWRITE].tv_sec +
955             ((double) tileRequest->tv[STARTWRITE].tv_usec / 1000000.0),
956             (double) tileRequest->tv[ENDWRITE].tv_sec +
957             ((double) tileRequest->tv[ENDWRITE].tv_usec / 1000000.0),
958             (double) tileRequest->tv[RECEIVED].tv_sec +
959             ((double) tileRequest->tv[RECEIVED].tv_usec / 1000000.0));
960         fwrite(logBuffer, sizeof(char), strlen(logBuffer), logFile);
961
962         if(!(readerStat->tilesRead % 50) && readerStat->tilesRead >= 50)
963         {
964         sprintf(logBuffer, "%f\tTiles Read = %d\tNet = %f"
965             "\tApp = %f\n",
966             (double) readerStat->time50[1].tv_sec +
967             ((double) readerStat->time50[1].tv_usec / 1000000.0),
968             readerStat->tilesRead,
969             tsm->issInfo.netAverage[proc],
970             tsm->issInfo.appAverage[proc]);
971         fwrite(logBuffer, sizeof(char),
972                 strlen(logBuffer), logFile);
973         }
974         if(flushFlag)
975         {
976         fflush(logFile);
977         }
978 }
979
980 /*
981  * Synopsis:  int TsReadDems(int)
982  *
983  *
984  * Description: Read the DEM files from the a filesystem. This is
985  *        a hack until we can get the TSM to send us the DEM's.
986  *        This function reads the DEM files corresponding to the
987  *        data set id passed in as a parameter and fills the tile tables
988  *        with the DEMs.
989  *
990  * Externals: TsTileStruct * tileStruct
991  *
992  *
993  * Returns: FALSE - On failure
994  *          TRUE  - On Success
995  *
996  * Author:
997  *      Stephen Lau
998  *
999  * Date:
1000 *
1001 */
1002 int
```

```
1003 TsReadDems(volatile TSMConnection * tsm)
1004 {
1005  static char * funcName = "TsReadDems()";
1006  TileSetTspec * tspecInfo;
1007  int i;
1008  int numLevels;
1009  char * config_val;
1010
1011     /* GET THE TSPEC INFORMATION */
1012     tspecInfo = TsGetTspecInfo(TsDemType);
1013     if(!tspecInfo)
1014     {
1015         my_status("%s: Unable to get tspec info for data set!",
1016           funcName);
1017     return FALSE;
1018     }
1019
1020     /* GET THE PATHNAME TO THE DEM FILES */
1021     if ((config_val = getconfig("DEM_PATHNAME")) == NULL)
1022     {
1023         my_status("%s: Unable to find config param: "
1024           "DEM_PATHNAME. Check the config file!",
1025           funcName);
1026         return FALSE;
1027     }
1028
1029     /* GET THE NUMBER OF DEM LEVELS FOR THIS DATA SET */
1030     numLevels = tspecInfo->maxLevel - tspecInfo->minLevel;
1031
1032     /* READ EACH LEVEL INTO THE TILE TABLES */
1033     for(i = 0; i <= numLevels; i++)
1034     {
1035 #ifdef DEBUG
1036         my_status("%s: Reading DEM Level %d out of %d",
1037           funcName, i, numLevels);
1038 #endif
1039     if(!TsReadDEMLevel(tsm, config_val,
1040                 i + tspecInfo->minLevel,
1041                 &tspecInfo->tspecLevel.tspecLevel_val[i]))
1042         {
1043         my_status("%s: Unable to read DEMs", funcName);
1044         return FALSE;
1045     }
1046     }
1047     return TRUE;
1048 }
1049
1050 /*
1051  * Synopsis: void TsDrawTileMgr(TsTileType);
1052  *
1053  *
1054  * Description: Simple function that draws the tile manager
1055  *          based on the type of data we are dealing with. The
1056  *          type is passed in as a parameter. The reason this
1057  *          is here, is because the actual tile managers are
1058  *          not visible external to this file.
1059  *
1060  * Externals: TsTileStruct * tileStruct
1061  *
1062  * Returns: NOTHING
1063  *
1064  * Author:
1065  *       Stephen Lau
1066  *
1067  * Date: 1994
1068  *
1069  */
```

```
2274     return &tileStruct->dataSetsInfo;
2275 }
2276
2277 /*
2278  * Synopsis:  int TsRequestDems(int)
2279  *
2280  *
2281  * Description: Requests the DEM tiles from the TSM.
2282  *
2283  * Externals: TsTileStruct * tileStruct
2284  *
2285  *
2286  * Returns: FALSE - On failure
2287  *          TRUE  - On Success
2288  *
2289  * Author:
2290  *       Stephen Lau
2291  *
2292  * Date: December 15, 1994
2293  *
2294  */
2295 int
2296 TsRequestDems(volatile TSMConnection * tsm)
2297 {
2298  static char * funcName = "TsRequestDems()";
2299  TileSetTspec * tspecInfo;
2300  int            i;
2301  int            numLevels;
2302  int            numRequests;
2303  RequestList *  requestList;
2304  int            errRet;
2305
2306 #ifdef DEBUG
2307     my_status("%s: Getting Type %d", funcName, TsDemType);
2308 #endif
2309     /* GET THE TSPEC INFORMATION */
2310     tspecInfo = TsGetTspecInfo(TsDemType);
2311     if(!tspecInfo)
2312     {
2313         my_status("%s: Unable to get tspec info for data set!",
2314           funcName);
2315     return -1;
2316     }
2317     numLevels = tspecInfo->maxLevel - tspecInfo->minLevel;
2318 #ifdef DEBUG
2319     my_status("%s: numLevels = %d\tMaxl = %d\tMin = %d",
2320           funcName,
2321           numLevels,
2322           tspecInfo->maxLevel,
2323           tspecInfo->minLevel);
2324 #endif
2325     numRequests = 0;
2326
2327     /* CHECK EACH LEVEL */
2328     for(i = 0; i <= numLevels; i++)
2329     {
2330 #ifdef DEBUG
2331     my_status("%s: Requesting level %d", funcName, i);
2332 #endif
2333         TsRequestDEMLevel(requestList,
2334               i + tspecInfo->minLevel,
2335               &tspecInfo->tspecLevel.tspecLevel_val[i],
2336               tileStruct->setsInfo.tileTables[TsDemType][i],
2337                       &numRequests,
2338               tsm);
2339     if(numRequests >= MAXNUMREQ)
2340     {
```

```
2341            break;
2342            }
2343        }
2344        if(numRequests > 0)
2345        {
2346        if(!tsmStopReqTiles(tsm->tsmh))
2347        {
2348            my_status("%s: Error when writing out"
2349                  " the request list!", funcName);
2350            exit(1);
2351        }
2352        }
2353        return numRequests;
2354    }
2355
2356    /*
2357     * Synopsis:  void TsRequestDEMLevel()
2358     *
2359     *
2360     * Description: We request a level until we max out the request list
2361     *
2362     *
2363     * Externals:
2364     *
2365     *
2366     * Returns: NOTHING
2367     *
2368     * Author:
2369     *      Stephen Lau
2370     *
2371     * Date:
2372     *    September 27, 1993
2373     *
2374     */
2375    static void
2376    TsRequestDEMLevel(RequestList *       requestList,
2377              int               level,
2378              SimpleTileSet     * tileSet,
2379              TsTile *            dem,
2380              int *               numRequests,
2381              volatile TSMConnection *     tsm)
2382    {
2383     static char * funcName = "TsRequestDEMLevel()";
2384     int x, y;
2385     int tileOffset = 0;
2386     int status;
2387     int priority = NOW;
2388     int errRet;
2389
2390        for(y = tileSet->minTile[1]; y <= tileSet->maxTile[1]; y++)
2391        {
2392        for(x = tileSet->minTile[0]; x <= tileSet->maxTile[0]; x++)
2393        {
2394            status = dem[tileOffset].flags;
2395            if(status != TS_RESIDENT)
2396            {
2397            dem[tileOffset].flags = TS_REQUESTED;
2398            if(!tsmReqTile(tsm->tsmh,
2399                       TSMGetCurrentGeoTspec(tsm, TsDemType),
2400                       x, y, level, priority))
2401            {
2402                my_status("%s: Error Writing Tile! %d",
2403                      funcName,
2404                      errRet);
2405                exit(1);
2406            }
2407            (*numRequests)++;
```

```
/*
 * Name: TerraVision.c
 *
 * Description: TerraVision: The code.
 *
 * Function List:
 *
 * Dependencies:
 *
 * Revision History:
 *      $Revision: 2.71 $
 *      $Date: 1996/04/27 11:52:12 $
 *      $Author: lau $
 *
 */
#include "TerraVision.h"

static char TerraVision_rcsid[] = "$Id: TerraVision.c,v 2.71 1996/04/27 11:52:12 lau ↙
    Exp $";
static char TerraVision_version[] = "$Revision: 2.71 $ $Date: 1996/04/27 11:52:12 $";

#ifdef AVS
#include <avs/avs.h>
#include <avs/field.h>
#include "vr_util.h"

int TerraVisionDesc();
static AVSfield *vr_input;
#endif

/* LOCAL FUNCTION PROTOTYPES */
static void  SigCb(int);
static XtAppContext     TerraVisionInitGraphics(int, char * []);
static int  TerraVisionSpawnProcs(void);
static void TerraVisionGraphicsLoop(XtAppContext);
static void TerraVisionCAVELoop(void);
static int  TerraVisionParseArgs(int, char * []);
static int  TerraVisionGetURLPaths(void);
static int  TerraVisionInitDataSet(volatile TSMConnection *);
static void TerraVisionCreateStatusBar(Widget, Widget);
static void TerraVisionQuit(void);
static void TerraVisionToggleChangedCB(Widget, XtPointer, XtPointer);
static void TerraVisionRender(void *);
static int  TerraVisionInitialize(void);
static int  TerraVisionInitialTSMConnect(void);
static void TerraVisionCreateMenuBar(Widget);
static Display * TerraVisionSetGLDisplay(char *);
static int TerraVisionCAVELoadData(void);
static void TerraVisionChangeGeoPyramid(void);

void  TSMPrepareDataSets(void *);

static XtTimerCallbackProc TerraVisionTimerCallback(XtPointer, XtIntervalId *);
#ifdef AVS
static XtTimerCallbackProc AVSTimerCallback(XtPointer, XtIntervalId *);
#endif

/* LOCAL VARIABLES */
static int logo_count = 0;

static XtIntervalId  widgetTimer;     /* Timer for misc widget functions */
#ifdef AVS
static XtIntervalId  avsTimer;        /* Timer for avs device */
#endif

volatile int demFlag = FALSE;
volatile int issDone = FALSE;
```

```
2077      ClockWidgetChangeTime(clockWidget);
2078      }
2079
2080      /* DO WE WANT TO DO A FRAMEGRAB AND SEND IT TO XCPS? */
2081      if(threeD->managed && xcpsFlag)
2082      {
2083      if(counter> 20)
2084      {
2085          /* GRAB THE FRAME */
2086          uspsema(threeD->grabber->imageLock);
2087          ThreeDWidgetFrameGrab(threeD,
2088                    (unsigned long *) threeD->grabber->imageData);
2089          X11FrameGrabberDisplayImage(threeD->grabber);
2090          counter = 0;
2091      }
2092      counter++;
2093      }
2094
2095      /* DO WE WANT TO DO A FRAMEGRAB AND SEND IT TO XCPS? */
2096      if(twoD->managed && xcpsFlag)
2097      {
2098      if(counter> 20)
2099      {
2100          /* GRAB THE FRAME */
2101          uspsema(twoD->grabber->imageLock);
2102          TwoDWidgetFrameGrab(twoD,
2103                  (unsigned long *) twoD->grabber->imageData);
2104          X11FrameGrabberDisplayImage(twoD->grabber);
2105          counter = 0;
2106      }
2107      counter++;
2108      }
2109      if(tileManager->managed)
2110      {
2111      TileManagerRedraw(tileManager);
2112      }
2113
2114      /* RESET THE TIMER */
2115      widgetTimer = XtAppAddTimeOut(appContext,
2116                    100,
2117                    (XtTimerCallbackProc)
2118                    TerraVisionTimerCallback,
2119                    appContext);
2120      return NULL;
2121  }
2122
2123  /*
2124   * Synopsis: static int TerraVisionSpawnProcs(void)
2125   *
2126   *
2127   * Description: Spawn the different threads.
2128   *
2129   *
2130   * Externals:
2131   *
2132   *
2133   * Returns: FALSE - on failure
2134   *          TRUE  - on success
2135   *
2136   * Author:
2137   *       Stephen Lau
2138   *
2139   * Date: 1993
2140   *
2141   */
2142  static int
2143  TerraVisionSpawnProcs(void)
```

```
2144 {
2145  static char * funcName = "TerraVisionSpawnProcs()";
2146  static unsigned long   clockResolution;
2147  char * config_val;
2148  int i;
2149
2150     /* FOR THE CAVE */
2151     if(cave)
2152     {
2153     /* WE WANT A VISIBILITY PROCESS FOR EACH WALL */
2154     for(i = 0; i < numActiveWalls; i++)
2155     {
2156 #ifdef DEBUG
2157         my_status("%s: Spawning %d",
2158               funcName,
2159               i);
2160 #endif
2161             if ((procPid[numProcs] = sproc(GenerateVisible,
2162                     PR_SALL, visible[i])) < 0)
2163             {
2164                 my_status("%s: Unable to spawn the tile visibilty process",
2165                   funcName);
2166             return FALSE;
2167             }
2168     }
2169         sprintf(procNames[numProcs++], "Visibility Generator");
2170
2171         /* WE ALSO WANT GATS */
2172 /*
2173     if(sproc(CAVEConnectToGATS, PR_SALL, NULL) < 0)
2174     {
2175         my_status("%s: Unable to connect to GATS.",
2176               funcName);
2177         return FALSE;
2178     }
2179         sprintf(procNames[numProcs++], "GATS Server");
2180 */
2181     }
2182     else
2183     {
2184     /* Process that generates the visible tiles */
2185 #ifdef DEBUG
2186     my_status("%s: Visible[0] = %d", funcName, visible[0]->cache);
2187 #endif
2188         if ((procPid[numProcs] = sproc(GenerateVisible,
2189                     PR_SALL, visible[0])) < 0)
2190         {
2191             my_status("%s: Unable to spawn the tile visibilty process",
2192               funcName);
2193     }
2194         sprintf(procNames[numProcs++], "Visibility Generator");
2195
2196         /* Process that does the rendering */
2197         if ((procPid[numProcs] = sproc(TerraVisionRender,
2198                                     PR_SALL, NULL)) < 0)
2199         {
2200             my_status("%s: Unable to spawn the rendering process",
2201                     funcName);
2202             return FALSE;
2203         }
2204         sprintf(procNames[numProcs++], "TerraVisionRender");
2205     }
2206 #ifdef DEBUG
2207     my_status("%s: Request[0] = %d", funcName, visible[0]->cache);
2208 #endif
2209     /* Generates the requests lists and ships them to the ISS */
2210     if ((procPid[numProcs] = sproc(TsMakeRequest,
```

```
2211                    PR_SALL, visible)) < 0)
2212     {
2213     my_status("%s: Unable to spawn the tile requestor process",
2214           funcName);
2215     return FALSE;
2216     }
2217     sprintf(procNames[numProcs++], "Request Generator");
2218
2219 #ifdef DEBUG
2220     /* Get the resolution of the clock from the config file */
2221     if ((config_val = getconfig("CLOCK_RESOLUTION")) == NULL)
2222     {
2223         my_status("%s: Unable to get the "
2224           "config value: CLOCK_RESOLUTION.\n"
2225           "Check the config file.",
2226           funcName);
2227     return FALSE;
2228     }
2229     clockResolution = atoi(config_val);
2230
2231     /* Clock for the system so everyone remains in sync */
2232     if ((procPid[numProcs] = sproc(Timer, PR_SALL, &clockResolution)) < 0)
2233     {
2234         my_status("%s: Unable to spawn the timer process",
2235           funcName);
2236     return FALSE;
2237     }
2238     sprintf(procNames[numProcs++], "Clock");
2239 #endif
2240
2241     return TRUE;
2242 }
2243
2244 /*
2245  * Synopsis: int TerraVisionInitDataSet(TSMConnection *)
2246  *
2247  *
2248  * Description: Initialize the tile tables,
2249  *           spawns of the reader threads,
2250  *           reads in the DEM's
2251  *
2252  * Externals:
2253  *
2254  *
2255  * Returns: TRUE - on success
2256  *          FALSE - on failure
2257  *
2258  * Author:
2259  *       Stephen Lau
2260  *
2261  * Date: 1995
2262  *
2263  *
2264  */
2265 static int
2266 TerraVisionInitDataSet(volatile TSMConnection * tsm)
2267 {
2268  static char * funcName = "TerraVisionInitDataSet()";
2269  char * config_val;
2270  int requestRateUsec, requestRateSec;
2271  int num;
2272
2273     /* CREATE TILE STRUCTURES */
2274     if (!TsCreateTileStruct(tsm, terraArena))
2275     {
2276         my_status("%s: Unable to create tile structs.",
2277           funcName);
```

```
2278            return FALSE;
2279        }
2280    #ifdef DEBUG
2281        my_status("%s: Requesting DEM Set", funcName);
2282    #endif
2283
2284        /* SPAWN OFF THE READER THREADS */
2285        if(!TSMSpawnThreads(tsm))
2286        {
2287        my_status("%s: Failed to spawn reader threads.",
2288              funcName);
2289        return FALSE;
2290        }
2291        /* DO WE WANT TO DO CHEESY LOCAL DEMS? */
2292        if(localDems)
2293        {
2294        /* READ DEMS */
2295        if(!TsReadDems(tsm))
2296        {
2297            my_status("%s: Unable to read the DEM files.",
2298                  funcName);
2299            return FALSE;
2300        }
2301        }
2302        else
2303        {
2304        /* GET THE TILE REQUEST RATE */
2305        if ((config_val = getconfig("REQUEST_RATE_USEC")) == NULL)
2306        {
2307            my_status("%s: Unable to get the config val: REQUEST_RATE_USEC\n"
2308                  "Check the config file",
2309                  funcName);
2310            exit(1);
2311        }
2312        requestRateUsec = atoi(config_val);
2313        if ((config_val = getconfig("REQUEST_RATE_SEC")) == NULL)
2314        {
2315            my_status("%s: Unable to get the config val: REQUEST_RATE_SEC\n"
2316                  "Check the config file",
2317                  funcName);
2318            exit(1);
2319        }
2320        requestRateSec = atoi(config_val);
2321
2322        /* REQUEST DEMS */
2323        while((num = TsRequestDems(tsm)) > 0)
2324        {
2325            my_status("%s: Requesting %d DEM's",
2326                  funcName, num);
2327            if(!WaitForAlarm(requestRateSec, requestRateUsec))
2328            {
2329            my_status("%s: Error with the alarm!",
2330                  funcName);
2331            exit(1);
2332            }
2333        }
2334    #ifdef DEBUG
2335        my_status("%s: Done Request Dems!!", funcName);
2336    #endif
2337        if(num < 0)
2338        {
2339            my_status("%s: Whoops!", funcName);
2340            return FALSE;
2341        }
2342        }
2343    #ifdef DEBUG
2344        my_status("%s: Done!", funcName);
```

```
#endif
    return TRUE;
}

/*
 * Synopsis: TSMPrepareDataSets(void *)
 *
 *
 * Description: Thread to prepare the data set
 *      selected for use. After prepartion,
 *      this thread dies
 *
 * Externals: demFlag - flag to load DEM's
 *        tsmPrep - flag to tell status of tsm
 *
 * Returns: NONE - this thread dies
 *
 * Author:
 *       Stephen Lau
 *
 * Date: 1996
 *
 */
void
TSMPrepareDataSets(void * threadData)
{
 static char * funcName = "TSMPrepareDataSets()";
 TSMConnection * tsm;
 char buffer[200];

    tsm = (TSMConnection *) threadData;
#ifdef DEBUG
    my_status("%s: Inititng data sets!", funcName);
#endif
    demFlag = TRUE;

    /* WE HAVE A NEW GEO PYRAMID */
    TerraVisionChangeGeoPyramid();

    /* INITIALZIE THE NEW DATA SET */
    if(!TerraVisionInitDataSet(tsm))
    {
    my_status("%s: Failed to initiialize data sets.",
          funcName);
    tsmPrep = CONNECTFAIL;
    return;
    }
#ifdef DEBUG
    my_status("%s: We're done!", funcName);
#endif
    tsmPrep = TRUE;
}

/*
 * Synopsis: int TerraVisionParseArgs(int, char * [])
 *
 *
 * Description: Parse the command line options
 *
 *
 * Externals:
 *       tsmConnect    - flag to auto connect to the TSM
 *       debugTexture - flag to use a "debug" texture
 *       noRequests    - flag to send no requests
 *       noISS         - flag for noISS
 *       noTexturePage- flag for paging textures
 *       logFlag       - flag for logging timing info to file
```

```
3283      /* TOGGLE TEXTURE CACHE LOGGING */
3284      if(w == menuTVLoggingTextureCacheButton)
3285      {
3286      if(flag)
3287      {
3288          if(!TVLoggerStartLogging(tvLogger,
3289                      TV_TEXTURE_CACHE_LOG))
3290          {
3291          my_status("%s: Unable to start tv logging %d",
3292                funcName,
3293                TV_TEXTURE_CACHE_LOG);
3294          }
3295      }
3296      else
3297      {
3298          if(!TVLoggerEndLogging(tvLogger,
3299                      TV_TEXTURE_CACHE_LOG))
3300          {
3301          my_status("%s: Unable to end tv logging %d",
3302                funcName,
3303                TV_TEXTURE_CACHE_LOG);
3304          }
3305          XmToggleButtonSetState(menuTVLoggingAllButton,
3306                      False, False);
3307      }
3308      }
3309
3310      /* TOGGLE PREDICTION LOGGING */
3311      if(w == menuTVLoggingPredictButton)
3312      {
3313      if(flag)
3314      {
3315          if(!TVLoggerStartLogging(tvLogger,
3316                        TV_PREDICT_LOG))
3317          {
3318          my_status("%s: Unable to start tv logging %d",
3319                funcName,
3320                TV_PREDICT_LOG);
3321          }
3322      }
3323      else
3324      {
3325          if(!TVLoggerEndLogging(tvLogger,
3326                      TV_PREDICT_LOG))
3327          {
3328          my_status("%s: Unable to end tv logging %d",
3329                funcName,
3330                TV_PREDICT_LOG);
3331          }
3332          XmToggleButtonSetState(menuTVLoggingAllButton,
3333                      False, False);
3334      }
3335      }
3336
3337      /* TOGGLE XCPS MODE, CURRENTLY DISABLED */
3338      if(w == menuXCPSButton)
3339      {
3340      *((int *) client_data) = flag;
3341      }
3342  }
3343
3344  /*
3345   * Synopsis: TerraVisionRender(void *)
3346   *
3347   *
3348   * Description: The top level function for
3349   *      the render thread. All the rendering
```

```
3350  *      for the views is done from this thread.
3351  *
3352  * Externals: twoD - the two d view
3353  *            threeD - the three d view
3354  *            renderLock - semaphore for locking
3355  *            the pipeline
3356  *
3357  * Returns: NONE
3358  *
3359  * Author:
3360  *      Stephen Lau
3361  *
3362  * Date: 1995
3363  *
3364  */
3365 static void
3366 TerraVisionRender(void * data)
3367 {
3368  static char * funcName = "TerraVisionRender()";
3369
3370     while(1)
3371     {
3372     uspsema(renderLock);
3373     if(twoD->managed)
3374     {
3375         TwoDWidgetDraw(twoD);
3376     }
3377     if(threeD->managed)
3378     {
3379         ThreeDWidgetDraw(threeD);
3380     }
3381     usvsema(renderLock);
3382     }
3383 }
3384
3385 /*
3386  * Synopsis: TerraVisionSetGLDisplay(char *)
3387  *
3388  *
3389  * Description: Gets the Display context based upon the
3390  *      name passed in as a string
3391  *
3392  *
3393  * Externals: none
3394  *
3395  *
3396  * Returns: The Display context, else NULL
3397  *
3398  * Author:
3399  *      Stephen Lau
3400  *
3401  * Date: June 1995
3402  *
3403  */
3404 static Display *
3405 TerraVisionSetGLDisplay(char * glDisplayName)
3406 {
3407  static char * funcName = "TerraVisionSetGLDisplay()";
3408  Display * display = NULL;
3409
3410    display = XOpenDisplay(glDisplayName);
3411    if(!display)
3412    {
3413     my_status("%s: Unable to open GL Display: %s",
3414           funcName, glDisplayName);
3415     return NULL;
3416    }
```

```
1  /*
2   * Name:  TsTsm.c
3   *
4   * Description: Interface to the TSM
5   *
6   * Function List:
7   *
8   * Dependencies:
9   *
10  * Revision History:
11  *         $Date: 1996/04/27 11:34:10 $
12  *         $Author: lau $
13  *
14  */
15 #include "TsPrivate.h"
16
17 static char TsTsm_c_rcsid[] = "$Header: /homedir/magic/software/CVS/TerraVision/
       libTileSvc/TsTsm.c,v 2.3 1996/04/27 11:34:10 lau Exp $";
18 static char TsTsm_c_version[] = "$Revision: 2.3 $";
19 static char TsTsm_c_date[] = "$Date: 1996/04/27 11:34:10 $";
20
21 static int  TSMGetGeoPyramids(volatile TSMConnection *, usptr_t *);
22 static int  TSMGetTileSets(volatile TSMConnection *, usptr_t *);
23 static int TSMISSInitialize(volatile TSMConnection *, usptr_t *);
24
25 #define ISS_DEFAULT_LOGFILE "/usr/tmp/tv_iss.log"
26
27 /*
28  * Synopsis: TSMConnection * CreateTSM(void)
29  *
30  *
31  * Description: Creates a new TSMConnection which is used in
32  *     communication with the TSM
33  *
34  *
35  * Externals: NONE
36  *
37  *
38  * Returns: pointer to new TSMConncetion on success
39  *      NULL on failure
40  *
41  * Author:
42  *       Stephen Lau
43  *
44  * Date: Feb 1995
45  *
46  */
47 volatile TSMConnection *
48 CreateTSM(usptr_t * terraArena)
49 {
50  static char * funcName = "CreateTSM()";
51  volatile TSMConnection * tsm;
52  char * config_val;
53
54     tsm = (volatile TSMConnection *) usmalloc(sizeof(TSMConnection),
55                              terraArena);
56     if(!tsm)
57     {
58     Status("%s: Error unable to allocate "
59            "space for TSM.", funcName);
60     return NULL;
61     }
62
63     /* GET THE DEFAULT TSM URL FROM THE CONFIG FILE */
64     if ((config_val = getconfig("TSM_URL")) == NULL)
65     {
66         Status("%s: Unable to get default TSM url "
```

```
603         Status("%s: Unable to get default ISS host "
604         "from the config value: ISS_HOSTNAME\nCheck the config "
605         "file.",
606            funcName);
607     return FALSE;
608     }
609     sprintf(tsm->issInfo.remoteHost,"%s",config_val);
610     if ((config_val = getconfig("ISS_LOGFILE")) == NULL)
611     {
612     tsm->issInfo.logFilename = strdup(ISS_DEFAULT_LOGFILE);
613     }
614     else
615     {
616     tsm->issInfo.logFilename = strdup(config_val);
617     }
618     tsm->issInfo.logging = FALSE;
619
620     tsm->issInfo.maxBursts =
621        (double *) usmalloc(sizeof(double) * 10,
622                terraArena);
623
624     if(!tsm->issInfo.maxBursts)
625     {
626     Status("%s: Unable to allocate max bursts",
627            funcName);
628     return FALSE;
629     }
630     tsm->issInfo.netAverage =
631        (double *) usmalloc(sizeof(double) * 10,
632                terraArena);
633     if(!tsm->issInfo.netAverage)
634     {
635     Status("%s: Unable to allocate net average",
636            funcName);
637     return FALSE;
638     }
639
640     tsm->issInfo.appAverage =
641        (double *) usmalloc(sizeof(double) * 10,
642                terraArena);
643     if(!tsm->issInfo.appAverage)
644     {
645     Status("%s: Unable to allocate app average",
646            funcName);
647     return FALSE;
648     }
649     for(i = 0; i < 10; i++)
650     {
651     tsm->issInfo.maxBursts[i] = 100000.0;
652     tsm->issInfo.netAverage[i] = 100000.0;
653     tsm->issInfo.appAverage[i] = 100000.0;
654     }
655     return TRUE;
656 }
657
658 /*
659  * Synopsis: int TSMSpawnThreads()
660  *
661  *
662  * Description: We spawn off the threads for the different
663  *    readers here.
664  *
665  *
666  * Externals:
667  *
668  *
669  * Returns: TRUE = success, FALSE - failure
```

```
 *
 * Author:
 *       Stephen Lau
 *
 * Date: Feb 1995
 *
 */
int
TSMSpawnThreads(volatile TSMConnection * tsm)
{
 static char * funcName = "TSMSpawnThreads()";
 extern volatile pid_t   procPid[];
 extern char procNames[20][50];
 extern volatile int numProcs;
 int i;
 int setIds[2];

    if((setIds[0] = TSMGetCurrentSetIdType(tsm, TsDemType)) < 0)
    {
    my_status("%s: Can't locate set id for Dems.",
          funcName);
    return FALSE;
    }
    if((setIds[1] = TSMGetCurrentSetIdType(tsm, TsOiType)) < 0)
    {
    my_status("%s: Can't locate set id for OIs.",
          funcName);
    return FALSE;
    }
#ifdef DEBUG
    my_status("%s: Before spawning %d readers. Asking for %d and %d",
          funcName,
          tsm->tsmh->numServers,
          setIds[0], setIds[1]);
#endif
    if(!tsmPrepareForDataSets(tsm->tsmh, setIds, 2))
    {
    my_status("%s: Unable to prepare data sets.",
          funcName);
    return FALSE;
    }
    my_status("%s: After spawning %d readers.",
          funcName,
          tsm->tsmh->numServers);

    tsm->issInfo.numServers = tsm->tsmh->numServers;

    /* SPAWN PROCESSES THAT READ FROM THE TSM */
    for(i = 0; i < tsm->tsmh->numServers; i++)
    {
    if((procPid[numProcs] =
        sproc(TsImageServerReader, PR_SALL, (void *) i))
       < 0)
    {
        my_status("%s: Unable to spawn reader threads.",
              funcName);
        return FALSE;
    }
    sprintf(procNames[numProcs++], "Tile Reader #%d", i + 1);
    }
    return TRUE;
}
```

```
1  /*
2   * Name:  VISIBLE.C
3   *
4   * Description: Generates visible tile lists and request tile lists.
5   *
6   * Function List:
7   *
8   * Dependencies:
9   *
10  * Revision History:
11  *      $Date: 1996/04/27 11:44:11 $
12  *      $Author: lau $
13  *
14  */
15 #include "Visible.h"
16
17 /* RCS INFO */
18 static char Visible_rcsid[] = "$Id: Visible.c,v 2.33 1996/04/27 11:44:11 lau Exp $";
19 static char Visible_version[] = "$Revision: 2.33 $ $Date: 1996/04/27 11:44:11 $";
20
21 static void BreadthFirstSearch(ListStruct *, ListStruct *, TsTileType);
22 static void KeepRequestRecord(IssHeader *, int);
23 static void GenerateRequests(volatile Visible *, ListStruct *);
24 static void GenerateAndSendRequests(volatile Visible *, int, int);
25 static void ResetISSHeader(IssHeader * issHeader);
26
27 /*
28  * Synopsis: VisibilityStruct * CreateVisible(ThreeDWidget *,
29  *                                            TwoDWidget *,
30  *                                            usptr_t *);
31  *
32  * Description:  Creates the Visible Structure which is used in
33  *     createing the visibility lists
34  *
35  * Externals:  NONE - Just say no to globals
36  *
37  * Returns: A pointer to the new visibility structure
38  *
39  * Author:
40  *      Stephen Lau
41  *
42  * Date: August, 1994
43  *
44  */
45 Visible *
46 CreateVisible(ThreeDWidget * threeDWidget,
47           TwoDWidget *   twoDWidget,
48           usptr_t *      terraArena)
49 {
50  static char * funcName = "CreateVisible()";
51  Visible * visible;
52
53     /* CREATE THE VISIBLE STRUCT */
54     visible = (Visible *) usmalloc(sizeof(Visible), terraArena);
55     if(!visible)
56     {
57         my_status("%s: Unable to allocate space "
58           "for visible struct!",
59           funcName);
60     return NULL;
61     }
62
63     visible->threeDWidget = threeDWidget;
64     visible->twoDWidget = twoDWidget;
65     visible->arena = terraArena;
66     return visible;
67 }
```

```
 68
 69 /*
 70  * Synopsis: void GenerateVisible(void *)
 71  *
 72  * Description: Generates a list of visible tiles dependent upon the
 73  *              state of the world.
 74  *
 75  * Externals:  NONE - just say no to globals
 76  *
 77  * Returns: NONE
 78  *
 79  * Author:
 80  *      Stephen Lau
 81  *
 82  * Date: 1993
 83  *
 84  */
 85 void
 86 GenerateVisible(void * data)
 87 {
 88  static char * funcName = "GenerateVisible()";
 89  volatile Visible * visible;
 90  struct timeval startTime, endTime;
 91  extern TVLogger * tvLogger;
 92  extern volatile int cave;
 93  int numTiles;
 94
 95     visible = (volatile Visible *) data;
 96     while(1)
 97     {
 98     uspsema(visibilityLock[visible->cache]);
 99
100     /* GET TIME FOR LOGGING */
101     gettimeofday(&startTime);
102
103         numTiles = -1;
104     if(!cave)
105     {
106         /* GENERATE THE VISIBLE LIST FOR THE OVERHEAD WIDGET */
107         if(visible->twoDWidget->managed)
108         {
109         numTiles = TwoDWidgetGenerateVisible(visible->twoDWidget);
110         }
111     }
112         /* GENERATE THE VISIBLE LIST FOR THE OUT THE WINDOW WIDGET */
113     if(visible->threeDWidget->managed &&
114        !visible->threeDWidget->threeDPanel->freezeTiles)
115     {
116         numTiles = ThreeDWidgetGenerateVisible(visible->threeDWidget,
117                             visible->cache);
118     }
119 #ifdef DEBUG
120     my_status("%s: numTile = %d", funcName, numTiles);
121 #endif
122     if(!cave)
123     {
124         if(numTiles != -1 && tvLogger->logging[TV_VISIBLE_LOG])
125         {
126         gettimeofday(&endTime);
127         TVLoggerLogVisibleData(tvLogger,
128                         &startTime,
129                         &endTime,
130                         numTiles);
131         }
132     }
133     usvsema(visibilityLock[visible->cache]);
134     }
```

```
135 }
136
137 /*
138  * Synopsis: void GenerateRequests(Visible *, ListStruct *)
139  *
140  * Description: Generates a Request list depending upon the state of the
141  *              world
142  *
143  * Externals: NONE - just say no to globals
144  *
145  * Returns: NONE
146  *
147  * Author:
148  *      Stephen Lau
149  *
150  * Date: 1993
151  *
152  */
153 void
154 GenerateRequests(volatile Visible *    visible,
155          ListStruct * requestList)
156 {
157  static char * funcName = "GenerateRequests()";
158  ListStruct workingList;
159  QuadTile * quadTree = NULL;
160  struct timeval startTime, endTime;
161  extern TVLogger * tvLogger;
162  extern volatile int cave;
163
164     /* GET TIME FOR LOGGING */
165     gettimeofday(&startTime);
166
167     /* RESET THE LIST */
168     ListZero(&workingList);
169
170     if(!cave)
171     {
172     /* GENERATE REQUESTS FOR THE OVERHEAD WIDGET */
173     if(visible->twoDWidget->managed)
174     {
175         quadTree = TwoDWidgetGenerateRequests(visible->twoDWidget);
176     }
177     }
178     /* GENERATE REQUESTS FOR THE THREE D WIDGET */
179     if(visible->threeDWidget->managed &&
180        !visible->threeDWidget->threeDPanel->freezeTiles)
181     {
182         quadTree = ThreeDWidgetGenerateRequests(visible->threeDWidget,
183                         visible->cache);
184     }
185     if(quadTree)
186     {
187     ListAdd(&workingList, quadTree);
188     BreadthFirstSearch(&workingList,
189                 requestList,
190                 TsOiType);
191     FreeQuadTree(quadTree);
192     }
193     if(tvLogger->logging[TV_PREDICT_LOG])
194     {
195         gettimeofday(&endTime);
196     TVLoggerLogPredictData(tvLogger,
197                     &startTime,
198                     &endTime,
199                     ListLength(requestList));
200     }
201 }
```

```
202
203 /*
204  * Synopsis:
205  *  void
206  *  TsMakeRequest(TsTileId *)
207  *
208  * Description: Thread that generates the request lists which
209  *    gets sent to the ISS. It loops and loops generating lists
210  *    based upon what is visible
211  *
212  * Externals:
213  *
214  * Returns:
215  *  NONE
216  *
217  */
218 void
219 TsMakeRequest(void * data)
220 {
221  static char * funcName = "TsMakeRequest()";
222  volatile Visible ** visible;
223  int requestRateUsec, requestRateSec;
224  char * config_val;
225  int i;
226  int errRet;
227  extern int numActiveWalls;
228  extern volatile int cave;
229
230     visible = (volatile Visible **) data;
231
232     /* GET THE REQUEST RATE */
233     if ((config_val = getconfig("REQUEST_RATE_USEC")) == NULL)
234     {
235         my_status("%s: Unable to get the config val: REQUEST_RATE_USEC\n"
236                   "Check the config file",
237                   funcName);
238         exit(1);
239     }
240     requestRateUsec = atoi(config_val);
241     if ((config_val = getconfig("REQUEST_RATE_SEC")) == NULL)
242     {
243         my_status("%s: Unable to get the config val: REQUEST_RATE_SEC\n"
244                   "Check the config file",
245                   funcName);
246         exit(1);
247     }
248     requestRateSec = atoi(config_val);
249
250     /* GENERATE THE REQUESTS */
251     while(1)
252     {
253         if(cave)
254     {
255         for(i = 0; i < numActiveWalls; i++)
256         {
257             GenerateAndSendRequests(visible[i],
258                   requestRateSec,
259                   requestRateUsec);
260         }
261     }
262     else
263     {
264 #ifdef DEBUG
265       my_status("%s: In here %d %d", funcName,
266                   requestRateUsec,
267                   requestRateSec);
268 #endif
```

```
269         GenerateAndSendRequests(visible[0],
270                     requestRateSec,
271                     requestRateUsec);
272     }
273     }
274 }
275
276 /*
277  * Synopsis: void GenerateAndSendRequests()
278  *
279  *
280  * Description: This function is the one that really
281  *    does the request list generation. It waits for
282  *    the alarm and then generates the request list which
283  *    then gets shipped to the TSM
284  *
285  * Externals:
286  *
287  *
288  * Returns: NOTHING
289  *
290  * Author:
291  *       Stephen Lau
292  *
293  * Date: 1993
294  *
295  */
296 static void
297 GenerateAndSendRequests(volatile Visible * visible,
298             int requestRateSec,
299             int requestRateUsec)
300 {
301  static char * funcName = "GenerateAndSendRequests()";
302  TileMgr * tileMgr;
303  int numRequests = 0;
304  TsTile * tile;
305  TsTileId * tileRequest;
306  ListStruct   visibleList;
307  int val;
308  float cutoff1;
309  float cutoff2;
310  int numObjs;
311  struct timeval startTime, endTime, currTime;
312  double requestTime;
313  double writeTime;
314  IssHeader issHeader;
315  int priority;
316  int errRet;
317  extern volatile TSMConnection * tsm;
318  extern TVLogger * tvLogger;
319
320     if(!WaitForAlarm(requestRateSec, requestRateUsec))
321     {
322     my_status("%s: Error with the alarm!",
323           funcName);
324     exit(1);
325     }
326     uspsema(requestLock);
327
328     ListZero(&visibleList);
329
330     /* GENERATE THE REQUEST LIST */
331     GenerateRequests(visible, &visibleList);
332
333     /* GET TIME FOR LOGGING */
334     gettimeofday(&startTime);
335
```

```
336     tileRequest = (TsTileId *) ListHead((ListStruct *) &visibleList);
337     numObjs = ListLength(&visibleList);
338 #ifdef DEBUG
339     my_status("%s: numRequests = %d", funcName, numObjs);
340 #endif
341     cutoff1 = numObjs;
342     cutoff2 = numObjs;
343     if(numObjs > 10)
344     {
345     cutoff1 = trunc(.5 * numObjs);
346     cutoff2 = trunc(.8 * numObjs);
347     }
348
349     numRequests = 0;
350     priority = NOW;
351     ResetISSHeader(&issHeader);
352
353     /* LOOP THROUGH LIST AND PROCESS IT */
354     while(tileRequest != NULL && numRequests < MAXNUMREQ)
355     {
356     tile = TsGetTile(tileRequest);
357     if(tile == (TsTile *) TS_UNAVAILABLE ||
358        tile == (TsTile *) TS_ABOVERES ||
359        tile == (TsTile *) TS_BELOWRES)
360     {
361         tileRequest = (TsTileId *)
362           ListNext((ListStruct *) &visibleList);
363         continue;
364     }
365     val = TsGetTileStatus(tileRequest->type,
366                   tileRequest->x,
367                   tileRequest->y,
368                   tileRequest->res);
369
370         /* IF WE DON'T ALREADY HAVE THE TILE, WE NEED TO REQUEST IT */
371     if(val != TS_RESIDENT)
372     {
373         tileRequest->setId = TSMGetCurrentSetIdType(tsm, TsOiType);
374 #ifdef DEBUG
375         my_status("%s: Requesting %d %d %d %d at priority %d",
376               funcName,
377               (int) tileRequest->x,
378               (int) tileRequest->y,
379               (int) tileRequest->res,
380               tileRequest->setId, priority);
381         if(tile)
382         {
383         my_status("%s: TileFlag =  %d",
384               funcName, (int) tile->flags);
385         }
386 #endif
387         tile->flags = TS_REQUESTED;
388         if(!tsmReqTile(tsm->tsmh,
389                TSMGetCurrentGeoTspec(tsm, TsOiType),
390                tileRequest->x,
391                tileRequest->y,
392                tileRequest->res, priority))
393         {
394             my_status("%s: Error Writing Tile! %d",
395                   funcName,
396                   errRet);
397             exit(1);
398         }
399         if(tvLogger->logging[TV_REQUEST_LOG])
400         {
401         gettimeofday(&currTime);
402         TVLoggerLogData(tvLogger,
```

```
403                 TV_REQUEST_LOG,
404                 TV_REQ_TILE,
405                 currTime.tv_sec,
406                 currTime.tv_usec,
407                 "%d; %d; %d; %d;",
408                 tileRequest->setId,
409                 tileRequest->x,
410                 tileRequest->y,
411                 tileRequest->res);
412         }
413 #ifdef DEBUG
414         my_status("%s: Requesting %d %d %d %d %d",
415               funcName,
416               tileRequest->x,
417               tileRequest->y,
418               tileRequest->res,
419               tileRequest->setId,
420               priority);
421 #endif
422         if(numRequests < cutoff1)
423         {
424         issHeader.numType0Requests++;
425         priority = NOW;
426         }
427
428         if(cutoff1 <= numRequests && numRequests < cutoff2)
429         {
430         issHeader.numType1Requests++;
431         priority = LATER;
432         }
433
434         if(cutoff2 <= numRequests)
435         {
436         issHeader.numType2Requests++;
437         priority = NEVER;
438         }
439         priority = NOW;
440         numRequests++;
441     }
442     else
443     {
444         /* OTHERWISE UPDATE THE TILE SAYING WE HAVE USED IT */
445         tileMgr = TsGetTileMgr(tileRequest->type);
446         TileMgrUpdateTile(tileMgr, tileRequest);
447     }
448
449     tileRequest = (TsTileId *) ListNext((ListStruct *) &visibleList);
450     }
451     if (numRequests > 0)
452     {
453     if(!tsmStopReqTiles(tsm->tsmh))
454     {
455         my_status("%s: Error when writing out"
456             " the request list!", funcName);
457         exit(1);
458     }
459     }
460     KeepRequestRecord(&issHeader, numRequests);
461 #ifdef DEBUG
462     my_status("%s: Num Requested = %d",
463           funcName,
464           numRequests);
465 #endif
466     if(tvLogger->logging[TV_REQUEST_LOG])
467     {
468     gettimeofday(&endTime);
469     TVLoggerLogRequestListData(tvLogger,
```

```
                    &startTime,
                    &endTime,
                    issHeader.numType0Requests,
                    issHeader.numType1Requests,
                    issHeader.numType2Requests);
    }
    ListFreeObjs(&visibleList);
    usvsema(requestLock);
}

/*
 * Synopsis: void KeepRequestRecord()
 *
 *
 * Description: Keeps track of the requests for bookeeping and
 *    performance measurement
 *
 *
 * Externals: RequestRecord  requestRecord
 *
 *
 * Returns: NOTHING
 *
 * Author:
 *      Stephen Lau
 *
 * Date: 1995
 *
 */
static void
KeepRequestRecord(IssHeader * header,
          int         numRequests)
{
 static char * funcName = "KeepRequestRecord()";
 int i, j;
 extern RequestRecord requestRecord;

    requestRecord.maxRequests = 0;
    for(i = 0; i < 49; i++)
    {
    for(j = 0; j < 4; j++)
    {
        requestRecord.record[j][i] = requestRecord.record[j][i + 1];
    }
    if (requestRecord.maxRequests < requestRecord.record[3][i + 1])
    {
        requestRecord.maxRequests = requestRecord.record[3][i + 1];
    }
    }
    requestRecord.record[0][i] = header->numType0Requests;
    requestRecord.record[1][i] = header->numType1Requests;
    requestRecord.record[2][i] = header->numType2Requests;
    requestRecord.record[3][i] = numRequests;
    if (requestRecord.maxRequests < numRequests)
    {
    requestRecord.maxRequests = numRequests;
    }
    requestRecord.amtRequested += numRequests * 128 * 128 * 3;
}

/*
 * Synopsis: void FreeQuadTree(QuadTile *)
 *
 * Description: Recursively destroys a quad tree structure
 *
 *
 * Externals: NONE - just say no to globals
```

```
537  *
538  * Returns: NONE
539  *
540  * Author:
541  *       Stephen Lau
542  *
543  * Date: 1993
544  *
545  */
546 void
547 FreeQuadTree(QuadTile * node)
548 {
549  static char * funcName = "FreeQuadTree()";
550  int i;
551
552     /* REACHED A LEAF SO RETURN */
553     if(!node)
554     {
555     return;
556     }
557
558     /* GO THROUGH THE LEAFS AND DESTROY THE SUB-TREES */
559     for(i = 0; i < 4; i++)
560     {
561     FreeQuadTree(node->child[i]);
562     node->child[i] = NULL;
563     }
564
565     /* DESTROY THE CURRENT NODE */
566     ListMemFree(node);
567     node = NULL;
568 }
569
570 /*
571  * Synopsis: QuadTile * CalcVisibility()
572  *
573  *
574  * Description: This function is called recursively
575  *    on all four children to determine if they are visible
576  *    or not. The end result is a complete quad tree
577  *
578  * Externals: NONE - Just say no to globals
579  *
580  *
581  * Returns: Pointer to QuadTree
582  *
583  * Author:
584  *       Stephen Lau
585  *
586  * Date:
587  *
588  */
589 QuadTile *
590 CalcVisibility(float        xStart,
591            float        yStart,
592            int          res,
593            void *       data,
594            VisibleFunc  visibleFunc,
595            SubdivideFunc    subdivideFunc)
596 {
597  static char * funcName = "CalcVisibility()";
598  float      midX, midY;
599  QuadTile * node;
600  int        i;
601  TsTileSetInfo * tileSetInfo;
602  TsDataSetInfo * dataSetInfo;
603
```

```
604     /* IF THIS TILE ISN"T VISIBLE, LEAVE */
605     if(!visibleFunc(xStart, yStart, res, data))
606     {
607     return NULL;
608     }
609     tileSetInfo = TsGetTileSetInfo(TsDemType);
610     dataSetInfo = TsGetDataSetInfo();
611     node = (QuadTile *) ListMemAlloc(sizeof(QuadTile));
612
613     node->x = (xStart - tileSetInfo->xStart) /
614               (dataSetInfo->tileWidth * power2[res]);
615     node->y = (yStart - tileSetInfo->yStart) /
616               (dataSetInfo->tileHeight * power2[res]);
617
618     node->res = res;
619     node->flag = NULL;
620     for(i = 0; i < 4; i++)
621     {
622         node->child[i] = NULL;
623     }
624 #ifdef DEBUG
625     my_status("%s: %d %d %d",
626           funcName, node->x, node->y, node->res);
627 #endif
628     res--;
629     if((res < 0) || !subdivideFunc(node, xStart, yStart, res, data))
630     {
631         node->flag |= TsOiLeaf;
632     return node;
633     }
634     midX = (dataSetInfo->tileWidth * power2[res+1]) / 2.0 + xStart;
635     midY = (dataSetInfo->tileHeight * power2[res+1]) / 2.0 + yStart;
636
637     /* CHECK THE CHILDREN */
638     node->child[0] = CalcVisibility(xStart, yStart, res,
639                       data, visibleFunc, subdivideFunc);
640     node->child[1] = CalcVisibility(midX, yStart, res,
641                     data, visibleFunc, subdivideFunc);
642     node->child[2] = CalcVisibility(xStart, midY, res,
643                       data, visibleFunc, subdivideFunc);
644     node->child[3] = CalcVisibility(midX, midY, res,
645                       data, visibleFunc, subdivideFunc);
646     return node;
647 }
648
649 /*
650  * Synopsis: void BreadthFirstSearch()
651  *
652  *
653  * Description: A breadthfirst search is done of the
654  *    srcList with the results ending up in the dest.
655  *    This orders the list for proper request order
656  *
657  * Externals: NONE - just say no to globals
658  *
659  *
660  * Returns: NOTHING
661  *
662  * Author:
663  *      Stephen Lau
664  *
665  * Date: 1993
666  *
667  */
668 static void
669 BreadthFirstSearch(ListStruct *srcList,
670             ListStruct *destList,
```

```
671             TsTileType  type)
672 {
673  static char * funcName = "BreadthFirstSearch()";
674  extern usptr_t * terraArena;
675  QuadTile * node = NULL;
676  TsTileId * tileId;
677  int i;
678  int val;
679  int tmpX, tmpY;
680
681     while((node = (QuadTile *) ListDelete(srcList)) != NULL)
682     {
683 #ifdef DEBUG
684     my_status("%s: Src List Length = %d",
685           funcName, ListLength(srcList));
686     my_status("%s: Dest List Length = %d",
687           funcName, ListLength(destList));
688 #endif
689     TsConvertOiTiles(node->x, node->y, node->res, &tmpX, &tmpY);
690     node->x = tmpX;
691     node->y = tmpY;
692
693 #ifdef DEBUG
694     my_status("%s: Searching = %d %d %d",
695           funcName,
696           node->x, node->y, node->res);
697 #endif
698     val = TsGetTileStatus(type,
699                   node->x,
700                   node->y,
701                   node->res);
702     if (val == TS_BELOWRES)
703     {
704         continue;
705     }
706     if(val != TS_ABOVERES)
707         {
708 #ifdef DEBUG
709         my_status("%s: Adding = %d %d %d",
710               funcName,
711               node->x, node->y, node->res);
712 #endif
713         tileId = (TsTileId *) ListMemAlloc(sizeof(TsTileId));
714         if(!tileId)
715         {
716             my_status("%s: Error in malloc!",
717               funcName);
718             TsCloseTileStruct(terraArena);
719         perror(NULL);
720             exit(0);
721         }
722         tileId->x = node->x;
723         tileId->y = node->y;
724         tileId->res = node->res;
725         tileId->type = type;
726
727         ListAppend(destList, tileId);
728     }
729     if (node->flag & TsOiLeaf)
730     {
731         continue;
732     }
733     for(i = 0; i < 4; i++)
734     {
735         if(node->child[i] != NULL)
736         {
737         ListAppend(srcList, node->child[i]);
```

```
738         }
739     }
740     }
741 }
742
743 /*
744  * Synopsis: VisibleStruct * ParseQuadTree()
745  *
746  *
747  * Description: We look for Leaves using this function.
748  *   We only want to find the leaves which are in memory
749  *   for the visible list
750  *
751  * Externals:
752  *
753  *
754  * Returns: A VisibleStruct * of the parsed quad tree
755  *
756  * Author:
757  *      Stephen Lau
758  *
759  * Date: 1993
760  *
761  */
762 VisibleStruct *
763 ParseQuadTree(ListStruct * parseList,
764           int           demMinRes,
765           int           demMaxRes,
766           int           oiMinRes,
767           int           oiMaxRes,
768           QuadTile *    node)
769 {
770  static char * funcName = "ParseQuadTree()";
771  int val;
772  int i;
773  int newX, newY, newX2, newY2;
774  VisibleStruct *visibleStruct;
775  VisibleStruct * children[4];
776  QuadTile* childNode;
777
778     /*
779      * IF POINTER IS NULL, NOTHING TO DO.
780      *
781      */
782     if(!node)
783     {
784     return (VisibleStruct *) TRUE;
785     }
786
787     /*
788      * LOOK FOR THE OI LEAVES
789      *
790      */
791     if (!(node->flag & (char) TsOiLeaf))
792     {
793     TsConvertOiTiles(node->x, node->y, node->res, &newX, &newY);
794     val = TsGetTileStatus(TsOiType, newX, newY, node->res);
795     if(val == TS_RESIDENT)
796     {
797         for(i = 0; i < 4; i++)
798         {
799         childNode = node->child[i];
800         if(!childNode)
801         {
802             continue;
803         }
804         TsConvertOiTiles(childNode->x,
```

```
805                 childNode->y,
806                 childNode->res, &newX2, &newY2);
807         val = TsGetTileStatus(TsOiType, newX2, newY2,
808                       childNode->res);
809         if(val != TS_UNAVAILABLE &&
810            val != TS_RESIDENT)
811         {
812             visibleStruct = (VisibleStruct *)
813                ListMemAlloc(sizeof(VisibleStruct));
814             visibleStruct->dem.x = node->x;
815             visibleStruct->dem.y = node->y;
816             visibleStruct->dem.res = node->res + 2;
817             visibleStruct->dem.type = TsDemType;
818
819             visibleStruct->oi.res = node->res;
820             visibleStruct->oi.x = (short) newX;
821             visibleStruct->oi.y = (short) newY;
822             visibleStruct->oi.type = TsOiType;
823
824             visibleStruct->prim = NULL;
825             return (VisibleStruct *) visibleStruct;
826         }
827         }
828     }
829     for(i = 0; i < 4; i++)
830     {
831        children[i] = ParseQuadTree(parseList, demMinRes, demMaxRes,
832                          oiMinRes, oiMaxRes, node->child[i]);
833     }
834     if(children[0] != NULL && children[1] != NULL &&
835        children[2] != NULL && children[3] != NULL)
836     {
837             for(i = 0; i < 4; i++)
838             {
839                 if(children[i] != (VisibleStruct *) TRUE)
840             {
841                 ListAdd(parseList, children[i]);
842             }
843             }
844         return (VisibleStruct *) TRUE;
845         }
846         for(i = 0; i < 4; i++)
847     {
848         if(children[i] && children[i] != (VisibleStruct *) TRUE)
849         {
850             ListMemFree(children[i]);
851         }
852         }
853     TsConvertOiTiles(node->x, node->y, node->res, &newX, &newY);
854     val = TsGetTileStatus(TsOiType, newX, newY, node->res);
855     if(val == TS_UNAVAILABLE)
856     {
857         return (VisibleStruct *) TRUE;
858     }
859     if(val == TS_RESIDENT)
860     {
861         visibleStruct = (VisibleStruct *)
862            ListMemAlloc(sizeof(VisibleStruct));
863         visibleStruct->dem.x = node->x;
864         visibleStruct->dem.y = node->y;
865         visibleStruct->dem.res = node->res + 2;
866         visibleStruct->dem.type = TsDemType;
867
868         visibleStruct->oi.res = node->res;
869         visibleStruct->oi.x = (short) newX;
870         visibleStruct->oi.y = (short) newY;
871         visibleStruct->oi.type = TsOiType;
```

```
872
873         visibleStruct->prim = NULL;
874         return (VisibleStruct *) visibleStruct;
875     }
876     return (VisibleStruct *) NULL;
877     }
878
879     /*
880      * OTHERWISE WE HAVE AN OI LEAF NODE.
881      *
882      */
883     TsConvertOiTiles(node->x, node->y, node->res, &newX, &newY);
884     val = TsGetTileStatus(TsOiType, newX, newY, node->res);
885
886     /*
887      * IF THE OI TILE IS NOT AVAILABLE, WE LEAVE THIS PART OF THE TREE
888      *
889      */
890     if(val == TS_UNAVAILABLE)
891     {
892     return (VisibleStruct *) TRUE;
893     }
894     if(val == TS_RESIDENT)
895     {
896     visibleStruct = (VisibleStruct *) ListMemAlloc(sizeof(VisibleStruct));
897         visibleStruct->dem.x = node->x;
898         visibleStruct->dem.y = node->y;
899         visibleStruct->dem.res = node->res + 2;
900         visibleStruct->dem.type = TsDemType;
901
902         visibleStruct->oi.res = node->res;
903     visibleStruct->oi.x = (short) newX;
904     visibleStruct->oi.y = (short) newY;
905         visibleStruct->oi.type = TsOiType;
906
907     visibleStruct->prim = NULL;
908     return visibleStruct;
909     }
910     return (VisibleStruct *) NULL;
911 }
912
913 /*
914  * Synopsis: void ResetISSHeader()
915  *
916  *
917  * Description: Reset the struct that keeps
918  *      track of requests
919  *
920  * Externals: NONE
921  *
922  *
923  * Returns: NOTHING
924  *
925  * Author:
926  *       Stephen Lau
927  *
928  * Date: 1994
929  *
930  */
931 static void
932 ResetISSHeader(IssHeader * issHeader)
933 {
934  static char * funcName = "ResetISSHeader()";
935  struct timeval rqstTime;
936
937     issHeader->numType0Requests = 0;
938     issHeader->numType1Requests = 0;
```

```
939     issHeader->numType2Requests = 0;
940
941     /* GET THE TIMESTAMP FOR THE REQUEST LIST  */
942     gettimeofday(&rqstTime);
943     issHeader->tv_sec = rqstTime.tv_sec;
944     issHeader->tv_usec = rqstTime.tv_usec;
945 }
946
```

```
/*****************************************************************************
 * Name:
 *      ThreeDWidget.c
 *
 * Description:
 *  Creation and manipulation of a three D widget
 *
 * Function List:
 *
 * Dependencies:
 *
 * Revision History:
 *      $Date: 1996/04/19 10:09:46 $
 *      $Author: lau $
 *
 *****************************************************************************/
#include "ThreeDWidget.h"
#include <bstring.h>
#include "Visible.h"

/* RCS INFO */
static char ThreeDWidget1_rcsid[] = "$Id: ThreeDWidget1.c,v 2.5 1996/04/19 10:09:46 lau Exp $";
static char ThreeDWidget1_version[] = "$Revision: 2.5 $ $Date: 1996/04/19 10:09:46 $";

/*#define OLDSUBDIV*/

extern int *blobby;

typedef enum
{
    CALC_VISIBILITY = 0,
    CALC_REQUESTS
}CalcVisibilityType;

/* STATIC LOCAL FUNCTIONS */
static void     ThreeDWidgetGenerateVisibleDem(ThreeDWidget *, int, int);
static double   GetAverageDemVal(double, double, int, int, ThreeDWidget *);
/* static double    BilinearInterpolate(double, double, double, double, double, double);*/
static double   GetWeightedElevation(double, double, double, ThreeDWidget *);
static double   GetOptimumLevel(ThreeDWidget *, double, int, int);
static double   GetOptimumOiLevel(ThreeDWidget *, double, short);
static QuadTileState    GetDemVisibleState(float *);
static short *pointerDepthVisit(int, int, int, ThreeDWidget *, volatile DepthVisit *);
static void CalcMinMax(double, double, double, double, ThreeDWidget *, int, int,
                double *, double *);
static int  ThreeDWidgetVisible(double, double, int, int, int, ThreeDWidget *, double *, Matrix, CalcVisibilityType, int);
static int  ThreeDWidgetSubdivide(QuadTile *, int, ThreeDWidget *, int, int, double, double, double);
static QuadTile *   ThreeDWidgetCalcVisibility(double, double, int, ThreeDWidget *, Matrix, CalcVisibilityType, int);
static double   RealPart(double);
static int  CalculateDemRes(double, double, double, double, ThreeDWidget *, int, int);
static void ThreeDWidgetCreateRenderPrimitive(ThreeDWidget *, View *, int, int);
static void ThreeDWidgetClearQuadList(ThreeDWidget *, ListStruct *);
static DemTileState * pointerDemVis(ThreeDWidget *, int, int, int, int);


#define BilinearInterpolate(val00, val10, val11, val01, xOffset,yOffset) ((1.0 - (yOffset)) * ((val00) * (1.0 - (xOffset)) + (val10) * (xOffset)) + (yOffset) * ( (val01) * (1.0 - (xOffset)) + (val11) * (xOffset)))

```

```
727     {
728         intLevel = tileSetInfo->maxLevel;
729     }
730     tmpElev[1] = GetDemVal(xVal, yVal, intLevel, threeDWidget);
731
732     elev = tmpElev[0] * (1.0 - levelWeight) + tmpElev[1] * levelWeight;
733     return elev;
734 }
735
736 /*****************************************************************************
737  * Synopsis:
738  *
739  *
740  * Description:
741  *
742  *
743  * Externals:
744  *
745  *
746  * Returns:
747  *
748  * Author:
749  *      Nat Bletter
750  *
751  * Date:
752  *
753  *****************************************************************************/
754 int
755 ThreeDWidgetGenerateVisible(ThreeDWidget * threeDWidget,
756                 int             cache)
757 {
758  static char * funcName = "ThreeDWidgetGenerateVisible()";
759  QuadTile *visibleTree = NULL;
760  VisibleStruct * visibleStruct;
761  TsTileSetInfo * demTileSetInfo, *oiTileSetInfo;
762  int level;
763  int numTiles;
764
765     ThreeDWidgetClearQuadList(threeDWidget,
766                     &threeDWidget->visibleList[cache][threeDWidget->currQuadList
       [cache]]);
767     threeDWidget->numTile = 0;
768
769     if(!threeDWidget->managed)
770     {
771     return 0;
772     }
773
774     demTileSetInfo = TsGetTileSetInfo(TsDemType);
775     oiTileSetInfo = TsGetTileSetInfo(TsOiType);
776
777     uspsema(threeDWidget->visibilityLock[cache]);
778
779 #ifdef DEBUG
780     my_status("%s: In GenerateVis!",
781           funcName);
782 #endif
783     /* clear out depth visit arrays to all 0's */
784     for (level = oiTileSetInfo->minLevel;
785      level <= oiTileSetInfo->maxLevel;
786      level++)
787     {
788 #ifdef DEBUG
789     my_status("%s: Zeroing Level %d and %d",
790           funcName,
791           level,
792           sizeof(short) *
```

```
793   threeDWidget->depth[cache][threeDWidget->currQuadList[cache]][level -
794         threeDWidget->oiMinLevel].numTile[0] *
795     threeDWidget->depth[cache][threeDWidget->currQuadList[cache]][level -
796         threeDWidget->oiMinLevel].numTile[1]);
797 #endif
798     if(threeDWidget->depth == NULL)
799     {
800         my_status("%s: Whoops trying to access nil",
801               funcName);
802         usvsema(threeDWidget->visibilityLock[cache]);
803         return 0;
804     }
805     bzero(threeDWidget->depth[cache][threeDWidget->currQuadList[cache]][level -
806                         threeDWidget->oiMinLevel].visited,
807           sizeof(short) *
808           threeDWidget->depth[cache][threeDWidget->currQuadList[cache]][level -
809                         threeDWidget->oiMinLevel].numTile[0] *
810           threeDWidget->depth[cache][threeDWidget->currQuadList[cache]][level -
811                           threeDWidget->oiMinLevel].numTile[1]);
812     }
813
814     /* SAVE THE CURRENT VIEW FOR THIS QUAD TREE */
815     memcpy(&threeDWidget->viewPoints[cache][threeDWidget->currQuadList[cache]],
816        &threeDWidget->view, sizeof(View));
817
818     ThreeDWidgetGenerateVisibleDem(threeDWidget, cache, threeDWidget->currQuadList
      [cache]);
819     visibleTree = ThreeDWidgetCalcVisibility(demTileSetInfo->xStart,
820                         demTileSetInfo->yStart,
821                         demTileSetInfo->maxLevel,
822                         (void *) threeDWidget,
823                         threeDWidget->viewMatrix[cache],
824                         CALC_VISIBILITY,
825                         cache);
826 #ifdef DEBUG
827     my_status("%s: Num = %d",
828           funcName, threeDWidget->numTile);
829     my_status("%s: Number of Visible Tiles = %d",
830           funcName, threeDWidget->numTile);
831 #endif
832
833     if(visibleTree)
834     {
835  #ifdef DEBUG
836     my_status("%s: Parsing Tree!", funcName);
837     my_status("%s: Before Num Tiles Visible = %d",
838           funcName,
839           ListLength(&threeDWidget->visibleList[cache][threeDWidget->currQuadList
      [cache]]));
840  #endif
841     visibleStruct =
842     ParseQuadTree(&threeDWidget->visibleList[cache][threeDWidget->currQuadList
      [cache]],
843               demTileSetInfo->minLevel,
844               demTileSetInfo->maxLevel,
845               oiTileSetInfo->minLevel,
846               oiTileSetInfo->maxLevel,
847               visibleTree);
848
849     if(visibleStruct && visibleStruct != (VisibleStruct *) TRUE)
850     {
851         ListAdd(&threeDWidget->visibleList[cache][threeDWidget->currQuadList[cache]],
852             visibleStruct);
853     }
854 #ifdef DEBUG
855     my_status("%s: After Num Tiles Visible = %d",
856           funcName,
```

```
857     ListLength(&threeDWidget->visibleList[cache][threeDWidget->currQuadList[cache]]))↙
        ;
858 #endif
859     FreeQuadTree(visibleTree);
860     ThreeDWidgetCreateRenderPrimitive(threeDWidget,
861               &threeDWidget->viewPoints[cache][threeDWidget->currQuadList[cache]],
862                       threeDWidget->currQuadList[cache], cache);
863     }
864     numTiles =
865        ListLength(&threeDWidget->visibleList[cache][threeDWidget->currQuadList      ↙
        [cache]]);
866
867     usvsema(threeDWidget->visibilityLock[cache]);
868
869     /* GET THE LOCK */
870     uspsema(threeDWidget->lock);
871
872     threeDWidget->nextDrawList[cache] = threeDWidget->currQuadList[cache];
873     threeDWidget->currQuadList[cache] = threeDWidget->nextQuadList[cache];
874     threeDWidget->nextQuadList[cache] = threeDWidget->nextDrawList[cache];
875 #ifdef DEBUG
876     my_status("%s: Q Done CR = %d NR = %d CQ = %d NQ = %d",
877           funcName,
878           threeDWidget->currDrawList[cache],
879           threeDWidget->nextDrawList[cache],
880           threeDWidget->currQuadList[cache],
881           threeDWidget->nextQuadList[cache]);
882 #endif
883     usvsema(threeDWidget->lock);
884     return numTiles;
885 }
886
887 /*
888  * Synopsis: ThreeDWidgetGenerateVisibleDem(ThreeDWidget * threeDWidget)
889  *
890  *
891  * Description: zeroes out 'computed' boolean of demVis array to set up
892  *     for sparse evaluation. Each cell is only computed when it is needed
893  *     since only a small amount (those in view) need to be evalutad
894  *     each frame.
895  *
896  * Externals:
897  *
898  *
899  * Returns:
900  *
901  * Author:
902  *       Nat Bletter
903  *
904  * Date: 1994
905  *
906  */
907 static void
908 ThreeDWidgetGenerateVisibleDem(ThreeDWidget * threeDWidget,
909                     int cache,
910                     int index)
911 {
912   TsTileSetInfo * tileSetInfo;
913   DemTileState *demVis;
914   int x,y;
915   int i;
916
917   tileSetInfo = TsGetTileSetInfo(TsDemType);
918
919   threeDWidget->refDemRes = tileSetInfo->maxLevel;
920   threeDWidget->refDemSize = tileSetInfo->tileWidth;
921
```

```
1849      return TRUE;
1850      }
1851
1852      divideThreshold = 128.0;
1853
1854  #ifdef OLDSUBDIV
1855      /* if tile is already less than 128 pixels (image size) don't subdivide */
1856      if(dist <= divideThreshold)
1857      {
1858      return FALSE;
1859      }
1860  #else
1861      minLevel = MAXFLOAT;
1862      xInc = tileSetInfo->tileWidth * power2[res];
1863      yInc = tileSetInfo->tileHeight * power2[res];
1864      for (x=xStart, i=0; i<2; i++, x+=xInc)
1865      {
1866         for (y=yStart, j=0; j<2; j++, y+=yInc)
1867         {
1868         if (((level = CalcOptimumOILevel(x, y, cache, index, threeDWidget))
1869          < minLevel) && (level>-1.0))
1870         {
1871             minLevel = level;
1872         }
1873         }
1874      }
1875      if (minLevel >= res)
1876      {
1877         return FALSE;
1878      }
1879  #endif
1880
1881      /*
1882       * IF THE CURRENT TILE IS WITHIN THE SUBDIVIDE THRESHOLD FOR OI TILES,
1883       * THEN THE NODE BECOMES AN OI LEAF
1884       *
1885       */
1886  #ifdef DEBUG
1887      my_status("%s: %d %f %f",
1888              funcName, res, dist, optimumLevel);
1889  #endif
1890      if((res + 1) <= tileSetInfo->minLevel)
1891      {
1892      return FALSE;
1893      }
1894      return TRUE;
1895  }
1896
1897  /*
1898   * Synopsis:
1899   *
1900   *
1901   * Description:
1902   *
1903   *
1904   * Externals:
1905   *
1906   *
1907   * Returns:
1908   *
1909   * Author:
1910   *       Stephen Lau
1911   *
1912   * Date:
1913   *
1914   */
1915  QuadTile *
```

```
1916 ThreeDWidgetGenerateRequests(ThreeDWidget * threeDWidget,
1917                   int cache)
1918 {
1919  QuadTile * quadTree = NULL;
1920  TsTileSetInfo * tileSetInfo;
1921
1922      if(!threeDWidget->managed)
1923      {
1924      return NULL;
1925      }
1926
1927      tileSetInfo = TsGetTileSetInfo(TsDemType);
1928      threeDWidget->numTile = 0;
1929      ThreeDWidgetGenerateVisibleDem(threeDWidget, cache, threeDWidget->currRequestList↙
      [cache]);
1930      quadTree = ThreeDWidgetCalcVisibility(tileSetInfo->xStart,
1931                        tileSetInfo->yStart,
1932                        tileSetInfo->maxLevel,
1933                        (void *) threeDWidget,
1934                        threeDWidget->bloatedViewMatrix[cache],
1935                        CALC_REQUESTS,
1936                           cache);
1937      return quadTree;
1938 }
1939
1940 /*
1941  * Synopsis:
1942  *
1943  *
1944  * Description:
1945  *
1946  *
1947  * Externals:
1948  *
1949  *
1950  * Returns:
1951  *
1952  * Author:
1953  *       Stephen Lau
1954  *
1955  * Date:
1956  *
1957  */
1958 static short *
1959 pointerDepthVisit(int res,
1960           int x,
1961           int y,
1962           ThreeDWidget *threeDWidget,
1963           volatile DepthVisit * depth)
1964 {
1965  static char * funcName = "pointerDepthVisit()";
1966  int level;
1967  int offset;
1968
1969 #ifdef DEBUG
1970      my_status("%s: In PointerDepth!",
1971           funcName);
1972 #endif
1973      level = res - threeDWidget->oiMinLevel;
1974 #ifdef DEBUG
1975      if (level < 0 ||
1976      x < depth[level].minTile[0] ||
1977      x > (depth[level].minTile[0] +
1978           depth[level].numTile[0]) ||
1979      y < depth[level].minTile[1] ||
1980      y > (depth[level].minTile[1] +
1981           depth[level].numTile[1]))
```

```
1982      {
1983      my_status("%s: level = %d, x = %ld, "
1984            "y = %ld, res = %ld, minlev = %ld, "
1985            "xmin = %ld xmax = %ld, ymin = %ld, ymax = %ld",
1986            funcName,
1987            level,
1988            x, y, res,
1989            threeDWidget->oiMinLevel,
1990            depth[level].minTile[0],
1991            depth[level].minTile[0] +
1992            depth[level].numTile[0],
1993            depth[level].minTile[1],
1994            depth[level].minTile[1] +
1995            depth[level].numTile[1]);
1996      }
1997  #endif
1998      offset = (x - depth[level].minTile[0]) +
1999          (y - depth[level].minTile[1]) *
2000        depth[level].numTile[0];
2001      if(offset >= blobby[level])
2002      {
2003  #ifdef DEBUG
2004      my_status("%s: Blobby! %d instead of %d at %d",
2005            funcName,
2006            offset,
2007            blobby[level],
2008            level);
2009  #endif
2010      return NULL;
2011      }
2012      return &depth[level].visited[offset];
2013
2014  }
2015
2016  /*
2017   * Synopsis: static QuadTile *
2018   *           ThreeDWidgetCalcVisibility(double, double, int, ThreeDWidget *, Matrix
      viewMatrix)
2019   *
2020   * Description: given tile position and resolution, return whether to cull
2021   *      it or not. passed normal or bloated matrix depending on whether it
2022   *      was called by generateRequests or generateVisible.
2023   *
2024   *
2025   * Externals:
2026   *
2027   *
2028   * Returns:
2029   *
2030   * Author:
2031   *      Stephen Lau
2032   *
2033   * Date:
2034   *
2035   */
2036  static QuadTile *
2037  ThreeDWidgetCalcVisibility(double       xStart,
2038                double       yStart,
2039                int          res,
2040                ThreeDWidget * threeDWidget,
2041                Matrix viewMatrix,
2042                CalcVisibilityType stat,
2043                int cache)
2044  {
2045   double     midX, midY;
2046   QuadTile * node;
2047   int        i;
```

```
2048  double     dist;
2049  TsTileSetInfo * tileSetInfo;
2050  TsDataSetInfo * dataSetInfo;
2051  int tileX, tileY;
2052  DepthVisit depth;
2053  short * tmp = NULL;
2054
2055      tileSetInfo = TsGetTileSetInfo(TsDemType);
2056      dataSetInfo = TsGetDataSetInfo();
2057
2058      tileX = (xStart - tileSetInfo->xStart) /
2059         (dataSetInfo->tileWidth * power2[res]);
2060      tileY = (yStart - tileSetInfo->yStart) /
2061         (dataSetInfo->tileHeight * power2[res]);
2062
2063  #ifdef DEBUG
2064      my_status("%s: Tn Calc!",
2065            funcName);
2066  #endif
2067
2068      if(stat != CALC_REQUESTS)
2069      {
2070      /* check if tile is out of bounds */
2071      depth = threeDWidget->depth[cache][threeDWidget->currQuadList[cache]][res -
      threeDWidget->oiMinLevel];
2072      if (tileX < depth.minTile[0] || tileX > depth.maxTile[0] ||
2073          tileY < depth.minTile[1] || tileY > depth.maxTile[1])
2074      {
2075          return NULL;
2076      }
2077      }
2078
2079      if(!ThreeDWidgetVisible(xStart, yStart, res, tileX, tileY,
2080                  threeDWidget, &dist, viewMatrix, stat, cache))
2081      {
2082      return NULL;
2083      }
2084
2085      node = (QuadTile *) ListMemAlloc(sizeof(QuadTile));
2086
2087      node->x = tileX;
2088      node->y = tileY;
2089      node->res = res;
2090      node->flag = NULL;
2091
2092      if(stat != CALC_REQUESTS)
2093      {
2094      /* mark that we've come down this far in depth visit array */
2095      tmp = pointerDepthVisit(node->res, node->x, node->y, threeDWidget,
2096                  threeDWidget->depth[cache][threeDWidget->currQuadList[cache]]);
2097      if(tmp)
2098      {
2099          *tmp = TRUE;
2100      }
2101  #ifdef DEBUG
2102      else
2103      {
2104          my_status("%s: Whoops nil tmp!",
2105                funcName);
2106      }
2107  #endif
2108      }
2109
2110      for(i = 0; i < 4; i++)
2111      {
2112      node->child[i] = NULL;
2113      }
```

```
2114
2115      threeDWidget->numTile++;
2116
2117      if(stat == CALC_VISIBILITY)
2118      {
2119      if(!ThreeDWidgetSubdivide(node,
2120                    res, threeDWidget,
2121                        cache,
2122                    threeDWidget->currQuadList[cache],
2123                    xStart, yStart, dist))
2124      {
2125          node->flag |= TsOiLeaf;
2126          return node;
2127      }
2128      }
2129      else
2130      {
2131      if(!ThreeDWidgetSubdivide(node,
2132                    res, threeDWidget,
2133                        cache,
2134                    threeDWidget->currRequestList[cache],
2135                    xStart, yStart, dist))
2136      {
2137          node->flag |= TsOiLeaf;
2138          return node;
2139      }
2140      }
2141   res--;
2142   midX = (dataSetInfo->tileWidth * power2[res+1]) / 2.0 + xStart;
2143   midY = (dataSetInfo->tileHeight * power2[res+1]) / 2.0 + yStart;
2144
2145   node->child[0] = ThreeDWidgetCalcVisibility(xStart,
2146                            yStart,
2147                            res,
2148                            threeDWidget,
2149                            viewMatrix,
2150                            stat,
2151                            cache);
2152
2153   node->child[1] = ThreeDWidgetCalcVisibility(midX,
2154                            yStart,
2155                            res,
2156                            threeDWidget,
2157                            viewMatrix,
2158                            stat,
2159                            cache);
2160
2161   node->child[2] = ThreeDWidgetCalcVisibility(xStart,
2162                            midY,
2163                            res,
2164                            threeDWidget,
2165                            viewMatrix,
2166                            stat,
2167                            cache);
2168
2169   node->child[3] = ThreeDWidgetCalcVisibility(midX,
2170                            midY,
2171                            res,
2172                            threeDWidget,
2173                            viewMatrix,
2174                            stat,
2175                            cache);
2176   return node;
2177 }
2178
2179 /*
2180  * Synopsis:
```

```
2515
2516 /*
2517  * Synopsis:
2518  *
2519  *
2520  * Description:
2521  *
2522  *
2523  * Externals:
2524  *
2525  *
2526  * Returns:
2527  *
2528  * Author:
2529  *      Stephen Lau
2530  *
2531  * Date:
2532  *
2533  */
2534 void
2535 ThreeDWidgetCreateUnclippedTexCoords(ThreeDWidget * threeDWidget)
2536 {
2537  double texXOffset, texYOffset;
2538  int length = 0;
2539  int x, y;
2540
2541     texXOffset = (256.0 / 256.0) / threeDWidget->numPolys;
2542     texYOffset = (256.0 / 256.0) / threeDWidget->numPolys;
2543
2544     threeDWidget->tunclipped[length][1] = 1 / 256.0;
2545     for(y = 0; y < threeDWidget->numPolys; y++)
2546     {
2547     threeDWidget->tunclipped[length][0] =
2548        texXOffset * threeDWidget->numPolys;
2549     for(x = 0; x < threeDWidget->numPolys + 1; x++)
2550     {
2551         length++;
2552         threeDWidget->tunclipped[length][1] =
2553            threeDWidget->tunclipped[length-1][1] + texYOffset;
2554         threeDWidget->tunclipped[length][0] =
2555            threeDWidget->tunclipped[length-1][0];
2556         length++;
2557         threeDWidget->tunclipped[length][1] =
2558            threeDWidget->tunclipped[length-2][1];
2559         threeDWidget->tunclipped[length][0] =
2560            threeDWidget->tunclipped[length-1][0] - texXOffset;
2561     }
2562     threeDWidget->tunclipped[length][1] =
2563        threeDWidget->tunclipped[length-1][1];
2564     }
2565 }
2566
2567 /*
2568  * Synopsis:
2569  *
2570  *
2571  * Description:
2572  *
2573  *
2574  * Externals:
2575  *
2576  *
2577  * Returns:
2578  *
2579  * Author:
2580  *      Stephen Lau
2581  *
```

```
2582  * Date: June 1995
2583  *
2584  */
2585 static void
2586 ThreeDWidgetCreateRenderPrimitive(ThreeDWidget * threeDWidget,
2587                   View * view,
2588                   int            listIndex,
2589                   int cache)
2590 {
2591  char * funcName = "ThreeDWidgetCreateRenderPrimitive()";
2592  float *tcoords;
2593  double realOiX, realOiY;
2594  int x, y;
2595  int maxX, maxY;
2596  float texclip[2];
2597  double xOffset, yOffset;
2598  double texXOffset, texYOffset;
2599  float *normal;
2600  float *normaltmp;
2601  float maxDataX, maxDataY;
2602  short neighbor[2][2];
2603  DepthVisit depthVisit;
2604  short * visit;
2605  float * v;
2606  int length, totlen;
2607  float pt[3], clipcoord[3];
2608  TsTileSetInfo * demTileSetInfo, * oiTileSetInfo;
2609  TsDataSetInfo * dataSetInfo;
2610  TileSetTspec *      tspecInfo;
2611  SimpleTileSet * tspecLevel;
2612  VisibleStruct * visibleStruct;
2613  TsTileId * visibleDem;
2614  TsTileId * visibleOi;
2615  RenderPrimitive *prim;
2616
2617     demTileSetInfo = TsGetTileSetInfo(TsDemType);
2618     oiTileSetInfo = TsGetTileSetInfo(TsOiType);
2619     dataSetInfo = TsGetDataSetInfo();
2620
2621     /* find the maximum ranges of actual data in set */
2622     tspecInfo = TsGetTspecInfo(TsOiType);
2623     tspecLevel =
2624        &tspecInfo->tspecLevel.tspecLevel_val[oiTileSetInfo->minLevel];
2625     maxDataX = tspecLevel->maxValidPixel[0] * power2[oiTileSetInfo->minLevel] +
2626        oiTileSetInfo->xStart;
2627     maxDataY = tspecLevel->maxValidPixel[1] * power2[oiTileSetInfo->minLevel] +
2628        oiTileSetInfo->yStart;
2629
2630     /* GET THE FIRST VISIBLE TILE FROM THE TILE LIST */
2631     visibleStruct = (VisibleStruct *)
2632        ListHead(&threeDWidget->visibleList[cache][listIndex]);
2633     while(visibleStruct != NULL)
2634     {
2635     /* GET THE OI AND DEM TILES FROM THE STRUCTURE */
2636     visibleOi = &visibleStruct->oi;
2637     visibleDem = &visibleStruct->dem;
2638
2639     if(visibleOi->res == INVALID_RES)
2640     {
2641         visibleStruct->prim = NULL;
2642
2643         /* GET THE NEXT TILE IN THE LIST */
2644         visibleStruct = (VisibleStruct *)
2645            ListNext(&threeDWidget->visibleList[cache][listIndex]);
2646
2647         continue;
2648     }
```

```
2649      if(!visibleStruct->prim)
2650      {
2651          visibleStruct->prim = prim = (RenderPrimitive *)
2652             usmalloc(sizeof(RenderPrimitive), threeDWidget->terraArena);
2653          if(!prim)
2654          {
2655          my_status("%s: Error allocating struct for prim",
2656                funcName);
2657          return;
2658          }
2659      }
2660      else
2661      {
2662          prim = visibleStruct->prim;
2663      }
2664          totlen = length = 0;
2665          prim->cPrims = 0;
2666      v = prim->p[0];
2667      tcoords = prim->tclipped[0];
2668      normal = prim->n[0];
2669      /*
2670       * CALCULATE THE REAL COORDINATES IN LCS OR UTM COORDINATES
2671       * BASED UPON THE TILE COORDINATES.
2672       */
2673      realOiX = (visibleOi->x *
2674             (oiTileSetInfo->tileWidth * power2[visibleOi->res]) +
2675             demTileSetInfo->xStart + dataSetInfo->oiOffset[0]);
2676
2677      realOiY = (visibleOi->y *
2678             (oiTileSetInfo->tileHeight * power2[visibleOi->res]) +
2679             demTileSetInfo->yStart + dataSetInfo->oiOffset[1]);
2680      /*
2681       * SUBDIVIDE THE POLYGON INTO THE DEM POSTINGS.
2682       *
2683       */
2684      if(visibleDem->res != INVALID_RES)
2685      {
2686          maxY =  power2[7 - (visibleDem->res - visibleOi->res)];
2687          maxX =  power2[7 - (visibleDem->res - visibleOi->res)];
2688      }
2689      else
2690      {
2691          maxY =  1;
2692          maxX =  1;
2693      }
2694      xOffset = ((oiTileSetInfo->tileWidth * power2[visibleOi->res]) /
2695             maxX);
2696      yOffset = ((oiTileSetInfo->tileHeight * power2[visibleOi->res]) /
2697             maxY);
2698
2699  #ifdef DEBUG
2700      my_status("%s: OiRes = %d Polys = %d",
2701            funcName,
2702            visibleOi->res,
2703            threeDWidget->numPolys);
2704  #endif
2705      xOffset *= maxX / threeDWidget->numPolys;
2706      yOffset *= maxY / threeDWidget->numPolys;
2707  #ifdef DEBUG
2708      my_status("%s: IN HERE!", funcName);
2709  #endif
2710      /* check if neighbors are at same resolution */
2711      /* if we're at the edge of dataset, pretend adjacent tile is at */
2712      /* same resolution */
2713      depthVisit =
2714         threeDWidget->depth[cache][listIndex][visibleOi->res -
2715                           threeDWidget->oiMinLevel];
```

```
2716      if (visibleOi->x <= depthVisit.minTile[0])
2717         neighbor[0][0] = TRUE;
2718      else
2719      {
2720          visit = pointerDepthVisit(visibleOi->res,
2721                        visibleOi->x-1,
2722                        visibleOi->y,
2723                        threeDWidget,
2724                        threeDWidget->depth[cache][listIndex]);
2725          if(visit)
2726          {
2727          neighbor[0][0] = *visit &&
2728             (TsCheckBoundary(TsOiType,
2729                      visibleOi->x-1,
2730                      visibleOi->y,
2731                      visibleOi->res) == TS_RESIDENT);
2732          }
2733          else
2734          {
2735          my_status("%s: Visit = NULL 4", funcName);
2736          neighbor[0][0] = FALSE;
2737          }
2738      }
2739      if (visibleOi->x >= depthVisit.maxTile[0])
2740         neighbor[0][1] = TRUE;
2741      else
2742      {
2743          visit = pointerDepthVisit(visibleOi->res,
2744                        visibleOi->x+1,
2745                        visibleOi->y,
2746                        threeDWidget,
2747                        threeDWidget->depth[cache][listIndex]);
2748          if(visit)
2749          {
2750          neighbor[0][1] = *visit &&
2751             (TsCheckBoundary(TsOiType,
2752                      visibleOi->x+1,
2753                      visibleOi->y,
2754                      visibleOi->res) == TS_RESIDENT);
2755          }
2756          else
2757          {
2758          my_status("%s: Visit = NULL 3", funcName);
2759          neighbor[0][1] = FALSE;
2760          }
2761      }
2762      if (visibleOi->y <= depthVisit.minTile[1])
2763         neighbor[1][0] = TRUE;
2764      else
2765      {
2766          visit = pointerDepthVisit(visibleOi->res,
2767                        visibleOi->x,
2768                        visibleOi->y-1,
2769                        threeDWidget,
2770                        threeDWidget->depth[cache][listIndex]);
2771
2772          if(visit)
2773          {
2774          neighbor[1][0] = *visit &&
2775             (TsCheckBoundary(TsOiType,
2776                      visibleOi->x,
2777                      visibleOi->y-1,
2778                      visibleOi->res) == TS_RESIDENT);
2779          }
2780          else
2781          {
2782          my_status("%s: Visit = NULL 2", funcName);
```

```
2783            neighbor[1][0] = FALSE;
2784            }
2785        }
2786        if (visibleOi->y >= depthVisit.maxTile[1])
2787        {
2788            neighbor[1][1] = TRUE;
2789        }
2790        else
2791        {
2792            visit = pointerDepthVisit(visibleOi->res,
2793                        visibleOi->x,
2794                        visibleOi->y+1,
2795                        threeDWidget,
2796                        threeDWidget->depth[cache][listIndex]);
2797            if(visit)
2798            {
2799            neighbor[1][1] = *visit &&
2800               (TsCheckBoundary(TsOiType,
2801                        visibleOi->x,
2802                        visibleOi->y+1,
2803                        visibleOi->res) == TS_RESIDENT);
2804            }
2805            else
2806            {
2807            my_status("%s: Visit = NULL 1", funcName);
2808            neighbor[1][1] = FALSE;
2809            }
2810        }
2811
2812        /* check if this tile is clipped, and figure out clipped coords */
2813        /* and tex coords */
2814        prim->clipped = FALSE;
2815        if ((pt[0] = realOiX + (xOffset * threeDWidget->numPolys)) > maxDataX)
2816        {
2817            clip(pt[0], maxDataX, &clipcoord[0], realOiX,
2818             1.0, &texclip[0], 0);
2819            prim->clipped = TRUE;
2820        }
2821        if ((pt[2] = realOiY + (yOffset * threeDWidget->numPolys)) > maxDataY)
2822        {
2823            clip(pt[2], maxDataY, &clipcoord[2], realOiY,
2824             1.0, &texclip[1], 0);
2825            prim->clipped = TRUE;
2826        }
2827
2828        pt[2] = realOiY;
2829
2830        for(y = 0; y < threeDWidget->numPolys; y++)
2831        {
2832            pt[0] = realOiX + (xOffset * threeDWidget->numPolys);
2833            for(x = 0; x < threeDWidget->numPolys + 1; x++)
2834            {
2835            if (prim->clipped)
2836            {
2837                if (pt[0] > maxDataX)
2838                {
2839                v[0] = clipcoord[0];
2840                tcoords[0] = texclip[0];
2841                }
2842                else
2843                {
2844                v[0] = pt[0];
2845                tcoords[0] =
2846                   threeDWidget->tunclipped[totlen+length][0];
2847                }
2848                if (pt[2] > maxDataY)
2849                {
```

```
2850                v[2] = clipcoord[2];
2851                tcoords[1] = texclip[1];
2852                }
2853                else
2854                {
2855                v[2] = pt[2];
2856                tcoords[1] =
2857                   threeDWidget->tunclipped[totlen+length][1];
2858                }
2859            }
2860            else
2861            {
2862                v[0] = pt[0];
2863                tcoords[0] = threeDWidget->tunclipped[totlen+length][0];
2864                v[2] = pt[2];
2865                tcoords[1] = threeDWidget->tunclipped[totlen+length][1];
2866            }
2867
2868            v[1] = GetAverageDemVal(v[0],
2869                        v[2],
2870                        cache,
2871                        listIndex,
2872                        threeDWidget);
2873
2874            if((x == threeDWidget->numPolys &&
2875                (y % 2 == 1) && !neighbor[0][0]) ||
2876               (x == 0 && (y % 2 == 1) && !neighbor[0][1]))
2877            {
2878                v[1] =
2879                   ((GetAverageDemVal(v[0],
2880                           (v[2] - yOffset),
2881                         cache,
2882                           listIndex,
2883                           threeDWidget) +
2884                    GetAverageDemVal(v[0],
2885                           (v[2] + yOffset),
2886                         cache,
2887                           listIndex,
2888                           threeDWidget)) / 2.0);
2889            }
2890
2891            if(y == 0 && x % 2 == 1 && !neighbor[1][0])
2892            {
2893                v[1] =
2894                   ((GetAverageDemVal((v[0] - xOffset),
2895                           v[2],
2896                         cache,
2897                           listIndex,
2898                           threeDWidget) +
2899                    GetAverageDemVal((v[0] + xOffset),
2900                           v[2],
2901                         cache,
2902                           listIndex,
2903                           threeDWidget)) / 2.0);
2904            }
2905
2906    #ifdef DEBUG
2907            my_status("%s: x = %f, y = %f, z = %f",
2908                  funcName,
2909                  v[0],
2910                  v[1],
2911                  v[2]);
2912    #endif
2913            normaltmp = (float *)
2914               TsGetDEMNormal(visibleDem,
2915                        31 - x * 32/threeDWidget->numPolys,
2916                        y * 32/threeDWidget->numPolys);
```

```
2917        normal[0] = normaltmp[0];
2918        normal[1] = normaltmp[2];
2919        normal[2] = normaltmp[1];
2920        normal += 3;
2921
2922        prim->p[totlen+length+1][0] = v[0];
2923        prim->tclipped[totlen+length+1][0] = tcoords[0];
2924        v[2] -= view->eye[2];
2925        v[0] -= view->eye[0];
2926        v[1] -= view->eye[1];
2927        v[0] *= -1;
2928
2929        /* increment to next vertex */
2930        length++;
2931        v += 3;
2932        tcoords += 2;
2933
2934        pt[2] += yOffset;
2935
2936        if (prim->clipped)
2937        {
2938            if (pt[2] > maxDataY)
2939            {
2940            v[2] = clipcoord[2];
2941            tcoords[1] = texclip[1];
2942            }
2943            else
2944            {
2945            v[2] = pt[2];
2946            tcoords[1] =
2947               threeDWidget->tunclipped[totlen+length][1];
2948            }
2949        }
2950        else
2951        {
2952            v[2] = pt[2];
2953            tcoords[1] = threeDWidget->tunclipped[totlen+length][1];
2954        }
2955
2956        v[1] = GetAverageDemVal(v[0],
2957                    v[2],
2958                    cache,
2959                    listIndex,
2960                    threeDWidget);
2961        if((x == threeDWidget->numPolys && (y + 1) % 2 == 1 &&
2962            !neighbor[0][0]) ||
2963           (x == 0 && (y + 1) % 2 == 1 && !neighbor[0][1]))
2964        {
2965            v[1] =
2966               ((GetAverageDemVal(v[0],
2967                       (v[2] - yOffset),
2968                    cache,
2969                      listIndex,
2970                      threeDWidget) +
2971             GetAverageDemVal(v[0],
2972                       (v[2] + yOffset),
2973                    cache,
2974                      listIndex,
2975                      threeDWidget)) / 2.0);
2976        }
2977
2978        if((y + 1) == threeDWidget->numPolys && x % 2 == 1 &&
2979           !neighbor[1][1])
2980        {
2981            v[1] =
2982               ((GetAverageDemVal((v[0] - xOffset),
2983                       v[2],
```

```
2984                    cache,
2985                      listIndex,
2986                      threeDWidget) +
2987             GetAverageDemVal((v[0] + xOffset),
2988                      v[2],
2989                    cache,
2990                      listIndex,
2991                      threeDWidget)) / 2.0);
2992         }
2993         v[2] -= view->eye[2];
2994         v[0] -= view->eye[0];
2995         v[1] -= view->eye[1];
2996         v[0] *= -1;
2997         normaltmp = (float *)
2998            TsGetDEMNormal(visibleDem,
2999                   31-x * 32/threeDWidget->numPolys,
3000                   (y + 1) * 32/threeDWidget->numPolys);
3001         normal[0] = normaltmp[0];
3002         normal[1] = normaltmp[2];
3003         normal[2] = normaltmp[1];
3004         normal += 3;
3005         /* increment to next vertex */
3006         length++;
3007         v += 3;
3008         tcoords += 2;
3009         pt[2] -= yOffset;
3010         pt[0] -= xOffset;
3011         }
3012         pt[2] += yOffset;
3013
3014         /* end of strip */
3015         prim->lengths[prim->cPrims] = length;
3016         prim->cPrims++;
3017         totlen += length;
3018         length = 0;
3019      }
3020      /* GET THE NEXT TILE IN THE LIST */
3021      visibleStruct = (VisibleStruct *)
3022         ListNext(&threeDWidget->visibleList[cache][listIndex]);
3023      }
3024  }
3025
3026  /*
3027   * Synopsis:
3028   *
3029   *
3030   * Description:
3031   *
3032   *
3033   * Externals:
3034   *
3035   *
3036   * Returns:
3037   *
3038   * Author:
3039   *       Stephen Lau
3040   *
3041   * Date: June 1995
3042   *
3043   */
3044  static void
3045  ThreeDWidgetClearQuadList(ThreeDWidget * threeDWidget,
3046                ListStruct * listStruct)
3047  {
3048   VisibleStruct * visibleStruct;
3049
3050      /* GET THE FIRST VISIBLE TILE FROM THE TILE LIST */
```