Programming Teaching Environment

Programming is becoming an essential area of knowledge as more and more human activities are being taken over by computers. As such there is a need to teach programming in schools to prepare students for this new work environment.
Project 1 Description

Continuing on the themes from Game Engine Concepts, one would like to add a user scripting system to bot Unity and Ogre to enable users to program objects that in a way that does not protects the core functionality of the game engine.

To protect the functionality of the engine the scripting language can only access to specific predefined functions that can only affect special objects with properties that allow them to interact with the scripting languages. Normally user avatars would not have such properties but bots probably would.
Additionally, one would want this scripting teaching environment to include the standard features of multi-player online games: networked servers, user authentication and user inventory.

A prototype system was developed last year for Unity.

See [http://classes.cs.kent.edu/gpg/trac/wiki/GamePracticumS14](http://classes.cs.kent.edu/gpg/trac/wiki/GamePracticumS14).

Extending that system or developing a similar open source system for Ogre would also be useful educational tool for testing teaching concepts.

Helpful tools Unity 5, Ogre, SmartFox or Raknet (see [http://classes.cs.kent.edu/gpg/trac/wiki/rbee](http://classes.cs.kent.edu/gpg/trac/wiki/rbee))
A visualization cluster is a computer cluster designed to do graphics in parallel. Various computationally and memory expensive tasks could be effectively handled by a visualization cluster.
Project 2: Visualization Clusters

Tiled Display - Data for display resides on machine that displays it.

Shared Data – Data required to produce display resides on several machines and must be transferred between machines to produce image. Simple example: 3D Game of life.
Project 2: Sharing Data
Volume Rendering

Wikipedia:
In scientific visualization and computer graphics, volume rendering is a set of techniques used to display a 2D projection of a 3D discretely sampled data set, typically a 3D array of data.

References
- Production Volume Rendering: Design and Implementation, on Safari
- OpenGL Development Cookbook, Chapter 7, on Safari
Project 2: Sharing Data

Similar neighborhood data sharing is required by
Cellular Automata
Graphics
Differential Equations
Image processing
Segmentation
Used in medicine, engineering, scientific research, etc.
Parallel Mesh Rendering and Segmentation

- **Image Segmentation (Wikipedia):**

  In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as superpixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

  The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).[1] When applied to a stack of images, typical in medical imaging, the resulting contours after image segmentation can be used to create 3D reconstructions with the help of interpolation algorithms like Marching cubes.
Project 2

Parallel Mesh Rendering and Segmentation
Sharing Data: MPI

Mpirun -np 4 - - hostfile lhosts myprog
-np 4 run four programs
lhosts file
Node0
Node1.cs.kent.edu
myprog a program using the mpi library
/* first.c */
#include <stdio.h>
#include <mpi.h>
int main(int argc, char ** argv) {
    int rank, size;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello world! I am %d of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```c
#include "mpi.h"
int main(argc, argv)
int argc;
char **argv;
{
    int rank, size, n, to, from, tagno;
    MPI_Status status;
    n = -1;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    to = rank + 1;
    if (rank == size -1) to = 0;
    from = rank - 1;
    tagno = 201;
    printf("Process %d of %d is alive\n",rank,size);
    if (rank == 0){
        from = size - 1;
        printf("Please enter a positive integer\n");
        scanf("%d",&n);
        printf("n = %d\n",n);
        MPI_Send(&n,1,MPI_INT,to,tagno,MPI_COMM_WORLD);
    }
    while (1){
        from = MPI_ANY_SOURCE;
        MPI_Recv(&n,1,MPI_INT,from,tagno,MPI_COMM_WORLD, &status);
        printf ("Rank %d received %d\n",rank, n);
        if (rank == 0) {n--;tagno++;}
        MPI_Send(&n,1,MPI_INT,to,tagno,MPI_COMM_WORLD);
        if (rank != 0) {n--;tagno++;}
        if (n<0){
            MPI_Finalize();
            return 0;
        }
    }
}
Compiling and Running

mpicc  mpicc first.c -o first

lhosts:
    node1
    node2

mpirun -np4 - -hostfile lhosts first

Problem: one must have special permission to access node1 and node2
Solution SSH

Create a folder .ssh in your main directory on strider.cs.kent.edu
Use ssh-keygen to generate an rsa key . Just accept default responses
Copy the contents of the file id_rsa.pub into .ssh/authorized_keys
After this you should be able to log into node1,node2, etc without a password.
MPI tutorials

There are many MPI tutorials available. Try googling “MPI tutorials”.
Lightly skim the tutorial mpi_guide.pdf
Run the examples above with 2, 4, 8 processes and 2, 4 nodes.
Find, compile and run a another simple mpi program.
Create a personal wiki page for yourself and add that program and its output to it.
Basic MPI Functions

int MPI_Comm_rank(MPI_Comm comm, int *rank)
int MPI_Comm_size(MPI_Comm comm, int *rank)
int MPI_Init(int *argc, char **argv)

-mpiquue: print out the state of the message queues when MPI_FINALIZE is called. All processors print; the output may be hard to decipher. This is intended as a debugging aid.

-mpiversion: print out the version of the implementation (not of MPI), including the arguments that were used with configure.

-mpinice nn: Increments the nice value by nn (lowering the priority of the program by nn). nn must be positive (except for root). Not all systems support this argument; those that do not will ignore it.

-mpeddbg: Start a debugger in an xterm window if there is an error (either detected by MPI or a normally fatal signal). This works only if MPICH was configured with -mpedbg. CURRENTLY DISABLED. If you have TotalView, -mpichtv or mpirun -tv will give you a better environment anyway.
int MPI_Init(int *argc, char **argv)

-mpimem : If MPICH was built with -DMPIR_DEBUG_MEM, this checks all malloc and free operations (internal to MPICH) for signs of injury to the memory allocation areas.

-mpidb options: Activate various debugging options. Some require that MPICH have been built with special options. These are intended for debugging MPICH, not for debugging user programs. The available options include

  mem      - Enable dynamic memory tracing of internal MPI objects
  memall   - Generate output of all memory allocation/deallocation
  ptr      - Enable tracing of internal MPI pointer conversions
  rank n   - Limit subsequent -mpidb options to on the process with the specified rank in MPI_COMM_WORLD. A rank of -1 selects all of MPI_COMM_WORLD.
  ref      - Trace use of internal MPI objects
  reffile filename - Trace use of internal MPI objects with output to the indicated file
  trace    - Trace routine calls
Basic MPI Functions

int MPI_Finalize()

Int MPI_Send(void * message, int count, MPI_Datatype datatype, int dest, int tag,
            MPI_Comm comm)

int MPI_Recv(void * message, int count ,MPI_Datatype datatype, int source, int tag,
             MPI_Comm comm, MPI_Status status)

<table>
<thead>
<tr>
<th>C datatype</th>
<th>MATLAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_datatype</td>
<td>C datatype</td>
</tr>
<tr>
<td>MPI_CHAR</td>
<td>signed char</td>
</tr>
<tr>
<td>MPI_SHORT</td>
<td>signed short int</td>
</tr>
<tr>
<td>MPI_INT</td>
<td>signed int</td>
</tr>
<tr>
<td>MPI_LONG</td>
<td>signed long int</td>
</tr>
<tr>
<td>MPI_UNSIGNED CHAR</td>
<td>unsigned char</td>
</tr>
<tr>
<td>MPI_UNSIGNED SHORT</td>
<td>unsigned short int</td>
</tr>
<tr>
<td>MPI_UNSIGNED INT</td>
<td>unsigned int</td>
</tr>
<tr>
<td>MPI_UNSIGNED LONG</td>
<td>unsigned long int</td>
</tr>
<tr>
<td>MPI_FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>MPI_DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>MPI_LONG DOUBLE</td>
<td>long double</td>
</tr>
<tr>
<td>MPI_BYTE</td>
<td></td>
</tr>
</tbody>
</table>
int MPI_Bcast(void * message, int count, MPI_Datatype datatype, int root, MPI_Comm comm)
### Basic MPI Commands

```c
int MPI_Reduce(void* operand, void* result, int count MPI_Datatype datatype, MPI_Op op,
               int root, MPI_Comm comm)
```

<table>
<thead>
<tr>
<th>Operation Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_MAX</td>
<td>Maximum</td>
</tr>
<tr>
<td>MPI_MIN</td>
<td>Minimum</td>
</tr>
<tr>
<td>MPI_SUM</td>
<td>Sum</td>
</tr>
<tr>
<td>MPI_PROD</td>
<td>Product</td>
</tr>
<tr>
<td>MPI_LAND</td>
<td>Logical And</td>
</tr>
<tr>
<td>MPI_BAND</td>
<td>Bitwise And</td>
</tr>
<tr>
<td>MPI_LOR</td>
<td>Logical Or</td>
</tr>
<tr>
<td>MPI_BOR</td>
<td>Bitwise Or</td>
</tr>
<tr>
<td>MPI_LXOR</td>
<td>Logical Exclusive Or</td>
</tr>
<tr>
<td>MPI_BXOR</td>
<td>Bitwise Exclusive Or</td>
</tr>
<tr>
<td>MPI_MAXLOC</td>
<td>Maximum and Location of Maximum</td>
</tr>
<tr>
<td>MPI_MINLOC</td>
<td>Minimum and Location of Minimum</td>
</tr>
</tbody>
</table>
Basic MPI Commands

```c
int MPI_Barrier(MPI_Comm comm);
```

**MPI_Barrier** provides a mechanism for synchronizing all the processes in the communicator `comm`. Each process blocks ie pauses until every process in `comm` has called **MPI_Barrier**.
Basic MPI Commands

```c
int MPI_Gather(void* send_buf, int send_count, MPI_Datatype send_type void * recv_buf, int recv_count, MPI_Datatype recv_type, int root MPI_comm comm)
```

Each process in comm sends the contents of `send_buf` to the process with rank `root`. The process `root` concatenates the received data in process rank order in `recv_buf`. That is the data from process is followed by the data from process 0 which is followed by the data from process 1 etc. The `recv` arguments are significant only on the process with rank `root`. The argument `recv_count` indicates the number of items received from each process -- not the total number received.
int MPI_Scatter( void* send_buf, int send_count,
        MPI_Datatype send_type, void* recv_buf,
        int recv_count, MPI_Datatype recv_type,
        int root, MPI_Comm comm)

The process with rank $\textbf{root}$ distributes the contents of $\textbf{send\_buf}$ among the processes. The contents of $\textbf{send\_buf}$ are split into $p$ segments each consisting of $\textbf{send\_count}$ items. The first segment goes to process 0 the second to process 1 etc. The $\textbf{send}$ arguments are significant only on process .
Basic MPI Commands

int MPI_Allgather(void* send_buf, int send_count,
                   MPI_Datatype send_type, void* recv_buf,
                   int recv_count, MPI_comm comm)

**MPI_Allgather** gathers the contents of each `send_buf` on each process. Its *effect* is the same as if there were a sequence of `p` calls to MPI_Gather, each of which has a different process acting as root.
int MPI_Allreduce(void* operand, void* result, 
    int count, MPI_Datatype datatype, 
    MPI_Op op, MPI_Comm comm)

MPI_Allreduce stores the result of the reduce operation 
op in each process result buffer.
Memory location that are used to store data from other nodes are called ghost points.

Will want $X(i+1,j-1,k+1)$ from other node
The ghost points should seem, as much as possible, like regular points to the underlying program.

$X(i+2,j-1,k+1)$ should reference the correct value.

While local indexing is correct, iterating through volume is more difficult.
Corner point is not (0,0,0). Could be (2,1,1).

C does not permit dynamic array declarations double X[I][J][K].

Can do self-indexing
\[ X(i,j,k) = X[(dimZ+2*bufZ)\times (dimY+2*bufY)\times (i-bufX) + (dimY+2*bufY)\times (j-bufY) + (k-bufZ)] \]
void GHOST_Obj_create(GHOST_Obj *P, int NumProcDims,
    int *proc_dims, int *bufsizes, int *wraps,int NumDataDims,
    int *data_dims, int ** d_list, MPI_Comm comm);
void GHOST_create_like(GHOST_Obj *P, GHOST_Obj Q);
int GHOST_Obj_free (GHOST_Obj *QQ);
void GHOST_Exchange_boundary(GHOST_Obj QQ);
void GHOST_Obj_Data_Fill(GHOST_Obj X, int size,double* b);
void GHOST_Obj_Data_Extract(GHOST_Obj X,
    int size,double* b);
The Tale of J. Random Newbie

Read of his woes

At least we have open source software. When there is a problem with the documentation, read the code.

The *The Art of Unix Programming* is worth reading (especially if you are also in system programming course)
Agile Software Development Methodologies

An iteration will be completed every three weeks. Each iteration involves a team working through a full software development cycle including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders (Muved an me). After each iterations on Monday the current group secretary will give a summary/demo of accomplishments, plans, and projections.
Agile Project Management

Characteristics

- Agile methods break tasks into small increments with minimal planning, and do not directly involve long-term planning. *Think top down design.*

- Each iteration (tri-weekly in our case) involves a team working through a full software development cycle including planning, requirements analysis, design, coding, unit testing, and acceptance testing when a working product is demonstrated to stakeholders. *It is an important part of the Agile to produce a working extension of previous iteration in each iteration.* Weekly team meetings (scrum) at the start of the week will discuss progress and plans for the week.

- No matter what development disciplines are required, each agile team will contain a customer representative. This person is appointed by stakeholders to act on their behalf and makes a personal commitment to being available for developers to answer mid-iteration problem-domain questions. *MuveED's Vice President (Steve Zapytowski) and I will be the company representatives.*
Each Friday the team will meet (a scrum) to jointly review the accomplishments from the prior week and lay tasks for the current week. Each assigned task will be submitted as ticket in TRAC.

In the team scrum on the first Friday of an iteration, the team will set milestones for the iteration and adjust as necessary the milestones (goals) for the project.

In the first team scrum the team will set milestones for the project.
Weekly Individual Planning/Progress Reports related to project. Due each Friday by 9:00 am.

- Reports must include
  - The list of tasks (with reference to corresponding tickets) assigned to you for the week.
  - The percentage of each task completed.
  - A summary discussion of what you did that week with links to associated tickets when you refer to an accomplishment or problem.
Writing Intensive Class Issues

Grading:

- Reports more than a week late will not be accepted without prior approval from the instructor.
- Graded on work accomplished, and
- Once or twice an iteration, a weekly report will be given a second grade based on clarity and correctness of exposition.
- You may correct an deficiency in clarity or exposition within a week for up to 95% of the original grade.
Writing Intensive Class Issues

- Group Planning/Progress Reports by current group secretary summarizing group progress.
  - A list of tickets assigned to each group member due on by 9:00 am each Monday.
  - A report on the progress made by the group due on the Friday after the end of a iteration by 9:00 am.
  - A verbal report and a demo showing the state of the project due on the Monday after an iteration ends.
Writing Intensive Class Issues

- Group Planning/Progress Reports by current group secretary summarizing group progress requirements:
  - In the verbal report: slides (for the stake holder) showing
    - A list of goals for the iteration, and corresponding tickets or milestones with percentages of completion attached to each goal.
    - A list of current project goals with its percentage completion indicated. In our project development paradigm, the project goals are refined each iteration by specifying more specific each iteration.
    - A demo showing the team's accomplishments.
  - Written report (for the boss) detailing all of the above, a list of the tickets assigned to each team member and an indication of its completion, and a summary of the progress made during the iteration (referencing tickets).
Writing Intensive Class Issues

- Group Planning/Progress Reports by current group secretary summarizing group progress.
  - Must include:
    - The prior iteration's lists of project goals.
    - The current list of project goals modified during the current iteration.
    - An estimate of the degree of completion of the current goals at the end of the iteration.
    - A link to the transportable code in the svn used to produce the demo.
Weekly reports: 40 % of grade.

Group reports: 20 % of grade.

Final Individual Report (at least 1000 words) and Final Group presentation (group demo required) 20% of grade. A report is required for each team member that should summarize his/her contribution to project (referencing tickets to support your statements) and indicate the relevance of that contribution. *In particular, the report should detail what part of the project documentation the student wrote.* This report will be used to assess you importance to the project.

Group Grade 10%. How well did your group perform. Regardless of how well your group does, if you are trying to make it work, you'll get a good grade. *Note that inadequate team project documentations will affect this mark.*

Class participation and attendance 10% of the grade.
Labs

- 243
- 355: new big servers.
- 136: Motion Capture.
Maxima and Octave

- **Maxima** is a computer algebra system that can solve virtually any equation that a high school or college student or can produce. It can solve many of the difficult problems of robotics.

- **Octave** is an numerical system that can solve virtually any equation that a high school or college student or can produce. It can give usable number solutions to many of the difficult problems of robotics.
Maxima and Octave

- Both systems are free and available for both linux and windows.
- Both can be connected to Unity using pipes.
- Or both could be connected to Unity though the web.
Unity 3D

- We will develop our system on Unity version 4.2.
- I will have a professional version available for in class use. You should install the free version on your home machines.
- A C# compiler is required. The suggest compiler is Visual Studio in addition to the compiler used by Unity.