CS 482/681: Simulations
Take-Home Final Exam (Dr. Lawlor)

0.) YOUR NAME: ___Answer J. Key I_____

Your finished electronic version of this exam document is due back by midnight 2015-05-07 as a PDF file. Please write clearly and professionally, using correct spelling and complete sentences, but keep things to the point. You are encouraged to use hyperlinks where appropriate. Everything you write should be your own work, in your own words.

This is an exam, but it is open web, open books, and open notes. You may use any non-living reference material, including any artificially intelligent entities; but classmates, friends, or faculty are not, unless you recreate them in a simulation from a nondestructive scan.

—

1.) Discretize this partial differential equation, which represents the motion of viscous water over a surface, so you can calculate the new value of a grid cell (at time t+dt) based on the values of the neighboring grid cells (with cell size h). Show both your mathematical reasoning, and your finished code.

$$\frac{\partial P}{\partial t} = P \frac{\partial P}{\partial x}$$

Recall that any derivative like dQ/dz can be approximated with a "finite difference" like:

dQ / dz ≈ change in Q / dz = (Q(z+dz) – Q(z)) / dz

Expanding finite differences here, we have:

dP/dt = P dP/dx

(P(x,t+dt) – P(x,t))/dt = P * (P(x+h,t)-P(x-h,t))/2h

Rearranging to solve for the new P value P(x,t+dt) gives:

P(x,t+dt) = P(x,t) + dt*P*(P(x+h,t)-P(x-h,t))/2h

new_P = P(x,t) + dt*P*(R-L)/2h

float P = old value of grid cell

float L,R,T,B = neighboring values in grid, each one a distance of h away from P

float new_P= P+dt*P*(R-L)/2h

2.a.) What is the minimum Courant-Friedrichs-Lewy (CFL) stable timestep for a gradient-based 4-neighbor Navier-Stokes fluid advection simulation, with fluid moving at 2 meters/second over a structured grid with cells of size 0.1 meters? Why?

CFL says the physical wavespeed must be less than the computational wavespeed. If we're using 4 neighbors (1-away neighbors) with a 0.1 meter grid, our computational wavespeed is one neighbor/step, or 0.1 meters/dt. CFL then means:

0.1 meters/dt > 2 meters/sec

Or rearranging to solve for dt,

0.1/2 sec > dt

0.05 sec > dt

This is our minimum timestep.

2.b.) To perform Navier-Stokes fluid advection in an unstructured tet mesh, what mesh data would need to be read or written, and what computation would need to be performed?

The Navier-Stokes fluid advection term is V dot grad V. So in any mesh, we just need a place to store V, such as at grid cell centers, and a way to compute grad V, such as by comparing values across the boundaries between grid cells. We then compute the dot product of these, to compute a velocity change.

Basic idea would be in each cell we compute the gradient as a finite difference:

```
vec3 gradV[3]=0.0; // gradients of V.x, V.y, V.z
for (int face=0;face<cell.face_count;face++)
for (int axis=0;axis<3;axis++) {
    float dV=cell.neighbor_by_face[face].V[axis] – cell.V[axis]; // delta V
    vec3 NdV = cell.face_normal[face]*dV; // delta V changes along face's normal vector
    gradV[axis]+=NdV/length(cell.neighbor_by_face[face].P-cell.P); // finite difference
}
vec3 VgV=vec3(dot(cell.V, gradV[0]), dot(cell.V, gradV[1]), dot(cell.V, gradV[2]));
V+=advection_speed*dt*VgV;
```

3.) Your research group is preparing a forest fire module for a whole-arctic ecosystem simulation, which stores these values at each simulation cell:

S: mass of **spruce** trees in cell

L: mass of large **leafy** (deciduous) trees in cell

B: mass of under**brush** in cell

M: mass of total ground-covering **moss**

H: average **humidity** of vegetation in cell (0 for bone-dry; 1 for soaking wet)

T: thickness of unfrozen **topsoil** in cell (meters)

F: thickness of perma**frost** layer (e.g., 0 for permafrost-free soil)

A: soil **acidity**, average in cell (pH scale)

W: proportion of surface area of cell covered by **water** (e.g., 1 for lake, 0.3 for bog)

3.a.) What are the top three most important variables changed by these events, and what change does the event cause on those variables?

Event	***Variable & Change 1***	***Variable & Change 2***	***Variable & Change 3***
Leaf fall	L-=leaf_mass();	T+=leaf_nutrients();	B-=leaf_smother();
Brush fire	B=0;	M=0;	H-=brushfire_energy(B);
Crown fire	S-=crown_mass(S);	L-=crown_mass(L);	H-=crown_energy(S,L);
Eutrophication	W-=eutroph(H,T);	B+=eutroph(H,T);	M+=eutroph(H,T);
Beaver damming	W+=dam(L,B);	B-=dam(0.0,B);	L-=dam(L,0.0);

3.b.) What are the top three variables that impact the probability a crown fire will start burning in a cell after a lightening strike, and what effect does each one have? What are the top three unlisted inputs you would need to compute this probability?

Listed variables: Humidity and Water inhibit crown fire, Spruce promotes crown fire.

Unlisted inputs: air temperature, windspeed, solar heating. (Runners up: topography)

3.c.) Would you recommend representing spruce trees using particles, a structured grid, an unstructured grid, or something else? Why?

Arguments could be made for all three. I'd make spruce mass one float in a structured grid, so I can run it on the GPU (and because I'm lazy).

4.) GPU simulations have been rightfully lauded as providing excellent performance, but this question discusses their limitations.

4.a.) How many values can be output by a pixel shader in WebGL? If there are more simulation values than this, how can we work around this limitation?

Your pixel shader can only write 4 color channels (RGBA). They default to 8-bit integer, but can be upgraded to 32-bit float if needed, allowing space for multiple small-bitsize inputs to be packed together into the remaining channels, although this is clunky to write and read. You can also take multiple passes, for example writing position to one texture, and then using a second shader to write velocity to a second texture, or interleaving even/odd pixels of the same shader.

4.b.) In many simulations, an interaction with a neighbor must often obey some invariant, for example when moving material around, the amount moved out of one pixel must equal the amount added into another. How can these invariants be enforced on the GPU?

There are two general solutions:

- Recompute the motion on both sides of the invariant. (Minimal memory, more computation)
- Compute the invariant motion once, write it out to a texture, and read it back from both sides that need it. (Minimal computation, more memory traffic)

4.c.) Is it possible to write GPU particle simulations using tree data structures? If so, how is the tree constructed?

Yes—we can walk through arbitrarily complex data structures on the GPU by doing repeated texture lookups (e.g., store texture coordinates in a texture pixel). You often build the data structure on the CPU, but you could probably do it using multiple passes on the GPU. A mipmap is essentially a spatial tree!