# Lecture 17

# Presentation Details

# Presentation Requirements (1 of 2)

- Presentation order: Decreasing order of team number
- Code cutoff is *this* Sunday at midnight
    - Have to ask permission to commit code afterwards, if your presentation depends on it
    - Spend time rehearsing instead

# Presentation Requirements (1 of 2)

- Readme files are now required inside Sunday's project submission.
  - Text summary of:
    - Non-obvious things implemented
    - Difficulties
    - Advanced topics used
    - Teammates' contributions

# Presentation Requirements (2 of 2)

- Teammate surveys due Friday of presentations
  - Submit a short text file assigning a score to each teammate, including a short comment about why

# Presentation Requirements (2 of 2)

- Plan how to set up for the projector quickly
- Find adapters for your computer!!!!
  - Otherwise there’s only my computer, navigated to your URL
  - My computer is slow
  - Your GitHub pages must be flipped on, otherwise even that won't work
  - Remember that enabling GitHub Pages is required
- If possible swing by this room
  - While it's still open, when no one is in here
  - Try plugging in.

# Grading

# Announcements

- Grading is late except for midterm
  - Don't panic about the impression that gives
  - Expect good grades in the class
  - ***Project grades will average high***
  - Final grades too
  - You earned it

# Course Evaluations (Required!)

# Announcement

- 2 free percentage points in the class from filling out a course evaluation
- (20 points on our 1000 point scale)
- This is another attempt to hit the 1100 points total required for A+ in the grading scale (that I promised would be used)

- Important:  Do not forget to fill one out!

# Final Exam Info and Tips

# Final Exam Format

- 40-50 multiple choice questions
- Scantron again
- Cumulative coverage (early stuff and late stuff)

- 11:30-2:30 Friday June 14
  - Young Hall CS76

# Final Exam Coverage

- The midterm’s harder questions:  Transformation ordering, moving between bases
- What's one sequence of matrices that can be used to connect two squares by a corner?
- Implicit, Explicit, and Parametric -- How do you convert between any pair of these equation types?
- Simple parametric shapes *f(s,t)*
- Hermite spline curves; G1 vs C1 continuity

# Final Exam Coverage

- What sequence of matrices reflects point P around an arbitrary plane, given the plane's equation?
- What sequence of matrices change a point's basis from any pair of arrows A to a pair B? (Be careful: Try a simple case first where A is just a single translation away from being B)
- What were the formulas used for ray tracing?
- What are the dependent variables when trying to find where a line's vanishing point is?

# Rule of Thumb

***Transforming a point** $P$**:***

Transformations: $T_1$, $T_2$, $T_3$

Matrix: $\mathbf{M} = \mathbf{M}_3\ \mathbf{M}_2\ \mathbf{M}_1$

Point is transformed by $\mathbf{M}P$

Each transformation happens with respect to the **same** coordinate system

***Transforming a coordinate system:***

Transformations: $T_1$, $T_2$, $T_3$ (not generally the same as the ones above)

Matrix: $\mathbf{M} = \mathbf{M}_1\ \mathbf{M}_2\ \mathbf{M}_3$

A point has coordinates $\mathbf{M}P$ in the original coordinate system

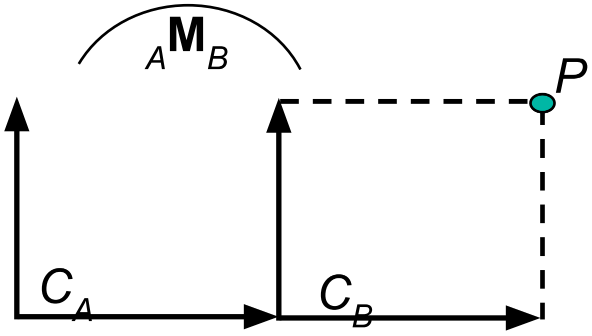
Each transformation happens with respect to **previous** coordinate system

# Rule of Thumb

**To find the transformation matrix that transforms P from $C_A$ coordinates to $C_B$ coordinates, we find a sequence of transformations that align $C_B$ to $C_{A,}$ accumulating matrices from left to right**

# Explanation of This Rule

Transformation **M**: ${}_A\mathbf{M}_B$




If we think coordinate systems, **M** takes $C_A$ from the left and produces $C_B$ on the right:

$$C_A \, {}_A\mathbf{M}_B = C_B$$

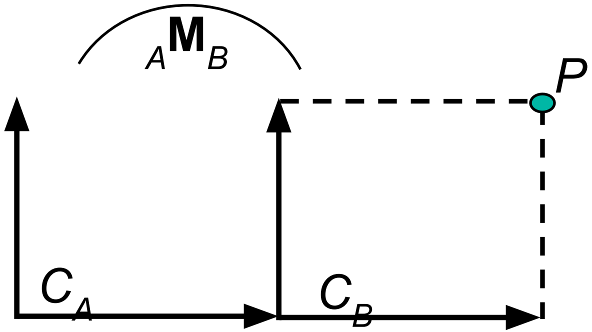
After this transformation we "talk" in $C_B$ coordinates (right side).

If we think points, then we go the other way; **M** takes $P_B$ on the right and produces the $P_A$ coordinates on the left:

$$P_A = {}_A\mathbf{M}_B \, P_B$$

# Explanation of This Rule

Transformation **M**: ${}_A\mathbf{M}_B$




Consider this simple example, where to produce $C_B$ we translate $C_A$ by +1 along the x axis:

$P_A = (2,1)$ $P_B = (1,1)$

If we move $C_A$ by +1 in *x* to transform it into $C_B$ then the *x* coordinate of *P* with respect to the new system is reduced by 1 ($C_B$ is closer to P than $C_A$ by 1).

So, if we want to transform the coordinates of *P* from $C_B$ to $C_A$ we need to add 1 in *x*. Exactly what we need to do to transform $C_A$ to $C_B$.

# Final Exam Coverage

- That’s all my tips
- Study the slides!  They’re the main source of questions.

# Prevent slowdown in your project: Performance profiling

# Using the Performance Profiler

- It’s one of the tabs in DevTools
- Switch to that tab and press the record button while your program is running
- Instantly receive a LOT of information about what parts were slowest
  - Top-down breakdown: Drill down into functions based on percentage of total runtime
- Another button can capture slowness during initial load (it records the page refresh)

# Preventing slowdown in graphics: Spatial Data Structures

# First problem:

- Problem #1:  You have (large #) of points.  Find all pairs that are close together.
  - This is a search problem
  - What are some CS32 data structures for fast lookup?

# Hash tables: "Spatial Hashing"

- We want some mapping of points to hash buckets:

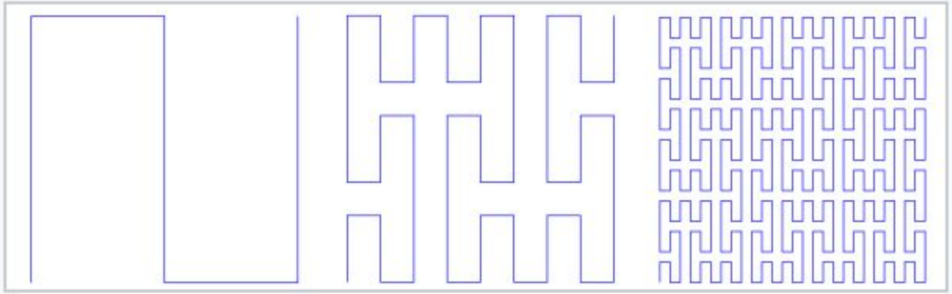
  f( x,y,z ) => 0 through 999

  (supposing 1000 buckets)
- Normally hash functions should be random
  - We'd lose neighborhoods that way
  - What we need is called "locality-sensitive hashing"
- Naive answer: Take floor of x,y,z then do:

  f( x,y,z ) = x + 10y + 100z.

# Hash tables: "Spatial Hashing"

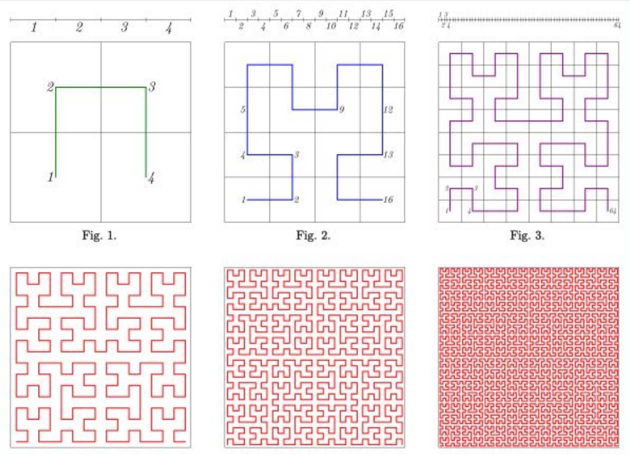
- Naive answer: Take floor of x,y,z then do:
  $f(x,y,z) = x + 10y + 100z$.
- Not cache-friendly; nearby objects one row away become far away in memory
- Better orderings: "Space-filling curve" order
  - Peano Curve: 1D curves that pass through every point in the unit square if you keep following them



Three iterations of the Peano curve construction, whose limit is a space-filling curve.

# Hash tables: "Spatial Hashing"

- Another one: Hilbert Curve




Six iterations of the Hilbert curve construction, whose limiting space-filling curve was devised by mathematician David Hilbert.

- Can generate a nice curve with "Morton Coding" (alternate the bits of the integers)

# Hash tables: "Spatial Hashing"

- For now suppose any ordering, such as f( x,y,z ) = x + 10y + 100z
  - Divides all of space into a 3D grid along integers
  - Does our scene have to be stuck in a 10x10x10 aquarium?
  - Let's try using x,y,z out of range anyway
  - Hopefully aliasing grid cells to same buckets results in an even distribution

# Next Problem:

- Problem #2: You have (large #) matrices, each of "bounding boxes" for objects we want to draw. Find all pairs that are close together.

- Naive solution: Do the same thing as for points
- Problem: Now our points have size
  - Some objects will dip over a grid cell's edge
  - Some objects will completely mismatch our grid scale

# Problem #2

- Some objects will dip over a grid cell's edge
- Some objects will completely mismatch our grid scale
- Solution 1: Be repetitive.
    - For each matrix, store a pointer to the object in *every* hash bucket (grid cell) that the object overlaps.

# Problem #2

- Problem: What if the object is huge and overlaps many or all of the grid cells?
    - We won't have reduced the search space at all
    - Objects that have movement / velocity become expensive
        - Remove every grid cell with the matching pointer (recursively scanning in all directions), add to new cells

# Problem #2

- Problem: What if the object is huge and overlaps many or all of the grid cells?
  - We won't have reduced the search space at all
  - Objects that have movement / velocity become very slow and expensive
    - Remove every grid cell with the matching pointer (recursively scanning in all directions), add to new cells

# Problem #2

- Solution 2:
  - Prevent mis-matched object scale and grid scale by maintaining a lot of grids.
    - Each grid is 2x the size of the previous
    - A bounding box is stored in the smallest grid where at most it overlaps 8 cells (in case of sitting on edges)

# Problem #2

- Implementation: Multiple hash tables, each with maybe 1000 hash buckets
    - An object is represented in one table, up to 8 times via pointers
- Software using this technqiue: "VDB" package of Houdini (Fluid simulations)
    - Video of someone using the plugin for tracking turbulence details within a smoke simulation: https://www.youtube.com/watch?v=MR8CA6BaVm8

# Benefits of our Spatial Hash Table

- “Frustum culling”
  - Reduces the # of draw calls
  - Easier to search for only the objects that are in view
- Collision detection
  - Now we reduced the problem from O(n^2) to O(1)
  - With a fairly small constant too, due to 8 neighbors max
- Ray tracing
  - We also reduced the ray / object collision problem down from O(n*pixels*recursions)

# Benefits of our Spatial Hash Table

- Ray tracing
  - We also reduced the ray / object collision problem down from O(n*pixels*recursions)
  - We still have to "march" the ray through the grid to the first struck object
  - Start at the biggest grid, have those cells pre-marked for if we need to drill down to a smaller grid
  - To march the ray (S,C), solve for t along the ray (S+tC) that first intersects a nearby grid plane (integers)

# Alternatives

- Tetrahedralized models
  - Instead of representing 2D surfaces (shells) of every object, fill in and connect the interior volume too
  - Make every object out of the simplest 3D shape, a tetrahedron
  - Common in elastics and fluids simulation (where forces are volumetric)

# Alternatives

- Tetrahedralized models
  - Main application for helping with spatial search:
    - **Tetrahedralize the air or empty space in between objects too**
  - Now everything is connected by edges
    - Edge connections always exist to tell you when objects about to collide
    - Ray tracing now just requires marching the ray across faces of neighboring tetrahedrons

# Alternatives

- Example software:  "Deformable Simplicial Complex"
  - Maintains this "mesh" even as objects move, by deforming tetrahedrons, adding & deleting when necessary
- Video 1:
  - My attempt (2D): https://www.youtube.com/watch?v=STlRbYcVY6E

# Alternatives

- Video 2:
  - Their attempt (3D): https://vimeo.com/48290450
    - Tetras in the air are not drawn (invisible)
      - They can freely deform/appear/disappear
    - Tetras touching a surface are hard constraints bound to the sim's forces
      - Notice the triangles popping in and out of existence when the sim deforms it

# What “real” WebGL programs can do

# “BioDigital” Web Human Anatomy Explorer

- Giant database of demos that run online and teach anatomy using WebGL models
- Knee example:
  - https://human.biodigital.com/widget/?m=production/maleAdult/musculoskeletal_system_knee.json

# “After the Flood” WebGL Demo

- Made with PlayCanvas Engine
- https://playcanv.as/e/p/44MRmJRU/
- A virtual scene that pushes performance limits