COMPSCI 777 S2 C 2004
Computer Games Technology

—A* Aesthetic Optimizations—

Hans W. Guesgen
Computer Science Department

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Motivation

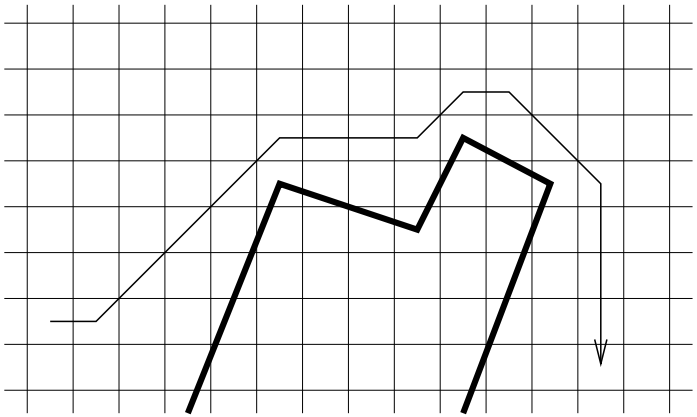
- Computing a path for a character in a game is more than just searching for the shortest path.
- It also involves creating an aesthetically pleasing path and resulting execution.
- Three ways of improving a path:
  - Make it straighter.
  - Make it smoother.
  - Make it more direct.

Straight Paths

- A* paths often look like being constructed by a drunk.
- They do not look natural in many situation.
- They undermine the believability of the game's AI.
- Two ways to fix the problem:
  - Promote straight paths within the A* algorithm.
  - Straighten the path afterwards.

Promoting Straight Paths within A*

- Promoting straighter paths involves careful cost weighting.
- Different paths often have the same costs:



Typical A* path

Straightened A* path

Promoting Straight Paths within A* (cont'd)

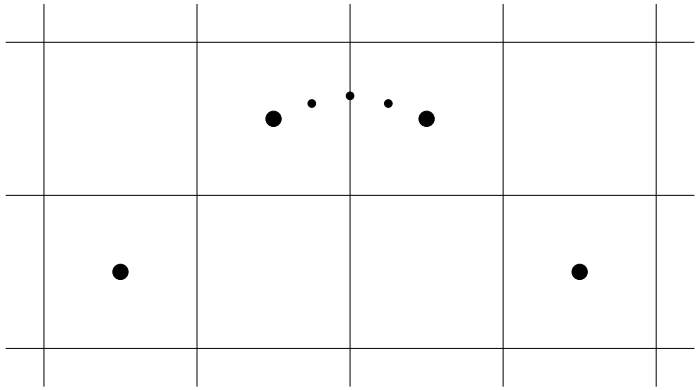
- A* is unable to differentiate between paths with identical costs.
- As a result, the choice of path is arbitrary.
- Factor in extra costs for not choosing a straight step.
- A reasonable penalty cost is half the normal cost of the step.
- In a regular grid, any penalty at all for non-straight choices does the trick.
- Penalizing non-straight paths may result in the search taking significantly more time.

Smooth Paths

- A* paths usually have sharp turns.
- Characters may look like moving robots.
- Making a path straight does not solve this problem.
- Applying rotational dampening to the turns masks them.
- However, the character "swings wide" as a result.
- A better way is to compute a Catmull-Rom spline.

Catmull-Rom Splines

- Unlike Bézier curves, Catmull-Rom splines keeps the control points.
- The Catmull-Rom formula requires four input points.
- It returns a smooth curve between the second and third points:



Code for the Catmull-Rom Formula

- The formula takes four points and a value u between 0 and 1 as input; it returns a new point:

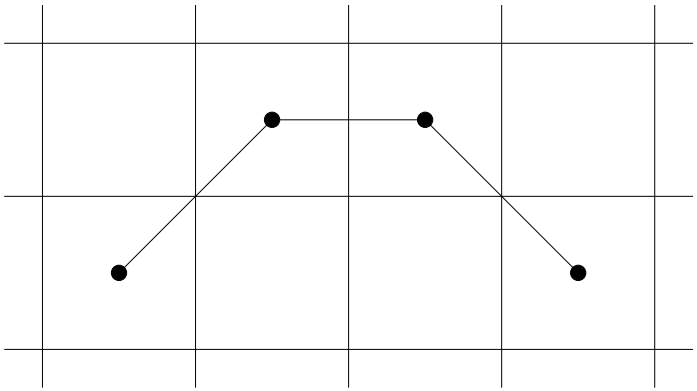
```
output_point   =   point_1 * (-0.5f*u*u*u + u*u - 0.5f*u) +
                   point_2 * (1.5f*u*u*u -2.5f*u*u + 1.0f) +
                   point_3 * (-1.5f*u*u*u + 2.0f*u*u + 0.5f*u) +
                   point_4 * (0.5f*u*u*u - 0.5f*u*u) +
```

- When u is 0, it gives you point_2.

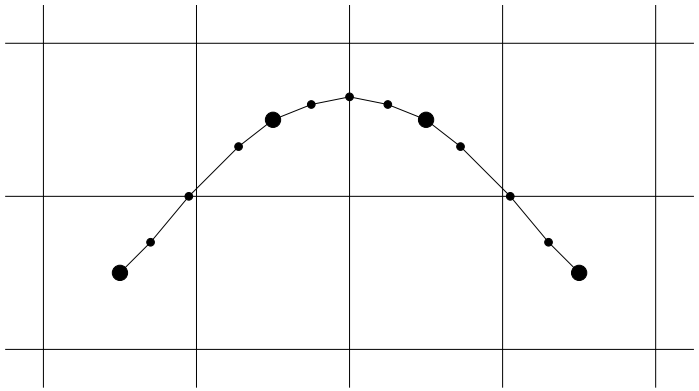
- When u is 1, it gives you point_3.

Applying the Catmull-Rom Formula

- To get the points between the first and second input points, give the function the first point twice, then the second and third.
- To get the points between the third and fourth, give the function the second and third, and double up on the fourth.



Original A* path



Smoothed A* path