3 Introduction to Computer Game Programming

Literature:

K. Besley et al.: Flash MX 2004 Games Most Wanted, Apress/Friends of ED 2004

Section 3.1 based on book chapter 1 by ***Glen Rhodes***

Source code for all examples at www.friendsofed.com

Why Computer Games in this Lecture?

- Computer game programming aims mostly at an immersive experience for the user
  - Same motivation is in the background for multimedia programming
- Game programming is a special case of multimedia programming
  - Uses highly interactive graphics, sound etc.
- Computer game programming is a huge market
  - But mostly ignored in university education
- Macromedia Flash is an excellent platform for computer game programming

Ingredients for a Good Computer Game

- Storytelling
  - Suspense
  - Humor
- Graphics
  - 2D
  - 3D
  - Animation
- Physics
  - Realistic behaviours
- Logic
  - Either logic not obvious to the player
    - » Player has to find the logic (e.g. when and where an object appears)
  - Or logic very simple and obvious
    - » Player has to find a counter-strategy

Logic Games

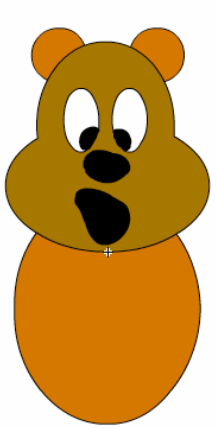
- A logic game may take 10 minutes to make and 20 minutes to play a round...
  - Simple rules, simply coded --> excellent gaming experience
  - Minimal set of logical rules can allow huge numbers of different games to be played
- Example: Rules for checkers
  - You move your pieces in a diagonal fashion, always forward.
  - If one of your pieces ends up diagonally adjacent to your opponent's piece, you may jump over his piece as long as your piece lands in an empty square. At this point, his piece is removed from the board.
  - You may take more than one of your opponent's pieces at a time if your diagonal jumps make this possible.
  - When your piece reaches the far end of the board, it is turned into a king, and then it earns the right to move diagonally forward and backward.
  - The game is over when you've taken all of your opponent's pieces or he's taken all of your pieces.

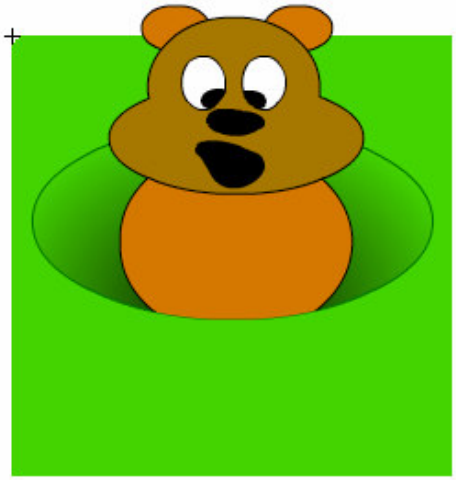
Example 1: Mole Invasion

- Goal: Get all the moles into their holes by clicking on them
- However: When a mole is clicked, all the other moles in adjacent holes on the left, right, top and bottom will toggle their positions.

Creating a Two-State Mole Symbol

Picture of
a single mole

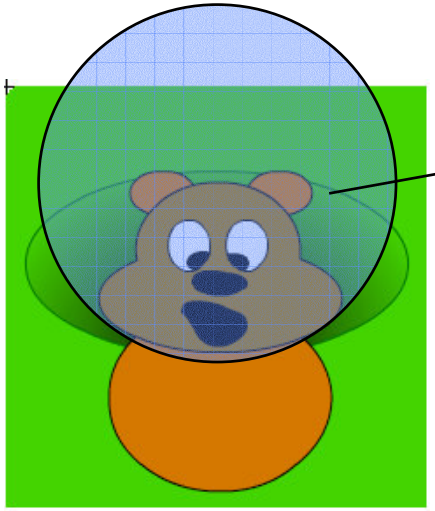




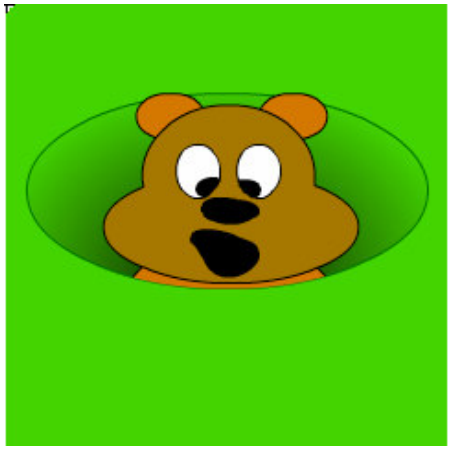
second
version:

mole
moved up

Mole in
hole

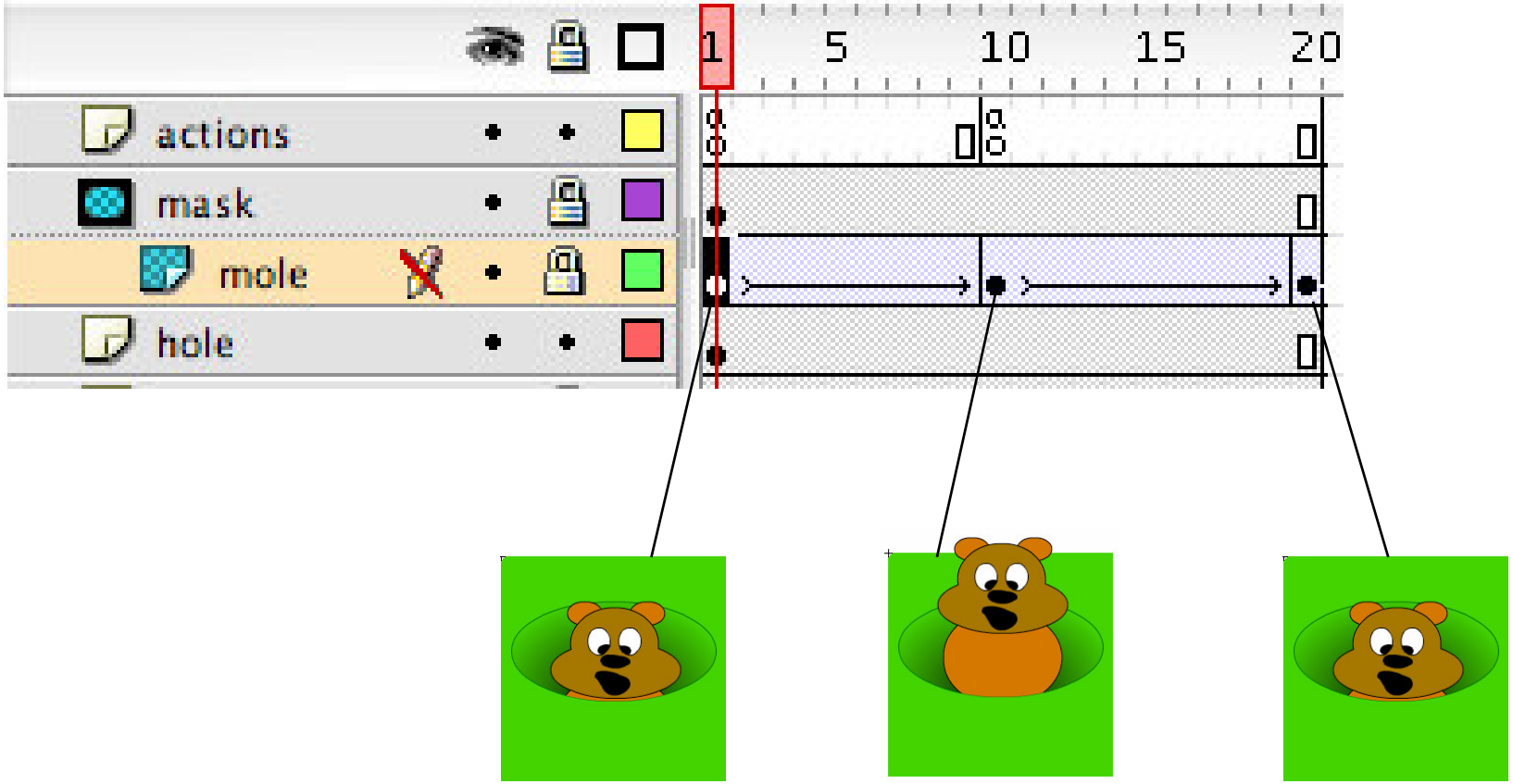


masking
the area where
the mole shall
appear



= background + mole picture

Animating the Mole Symbol



Toggling the mole position:
```
mole.play();
```
Moving a mole up:
```
mole.gotoAndStop(10);
```

Game Initialization

```
COLUMNS = 11;
ROWS = 8;
TILE_X = 60;
TILE_Y = 60;
TILE_Y_OFFSET = 10;

init();

function init() {
  var moleCount = 0;
  for (var i = 0; i < COLUMNS; i++) {
      for (var j = 0; j < ROWS; j++) {
            var h = this.attachMovie
              ("hole", ("hole_" + i + "_" + j), moleCount++);
            h._x = i * TILE_X;
            h._y = j * TILE_Y + TILE_Y_OFFSET;
            h.column = i;
            h.row = j;
            if (Math.random() > .5) h.gotoAndStop(10);
      }
  }
}
```

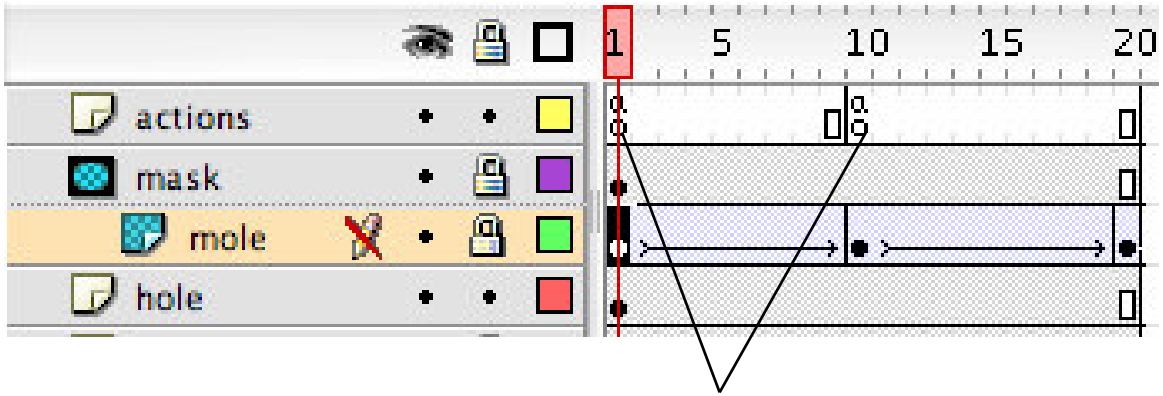
MovieClip.attachMovie()

- **`mc.attachMovie(libName, newId, depth)`**
  - Purpose: Dynamically create within ActionScript code new instances of symbols
  - Method; attaches a movie clip symbol from the library to a movie clip *mc* visible on the stage
  - Creates a new instance of the symbol (similar to “new”)
  - *libName* is the name given to the symbol in the library
  - *newId* is the identifier for the new instance
  - *depth* defines the z-order value for the new instance

- Attached movie clips can be removed again with the method **`removeMovieClip()`**

Event Handlers

```
function moleClick() {
  this.play();
  this._parent["hole_" + (this.column-1) + "_" +
    (this.row)].play();
  this._parent["hole_" + (this.column+1) + "_" +
    (this.row)].play();
  this._parent["hole_" + (this.column) + "_" +
    (this.row-1)].play();
  this._parent["hole_" + (this.column) + "_" +
    (this.row+1)].play();
  delete this.onRelease;
}
```




```
this.onRelease = this._parent.moleClick;
stop();
```

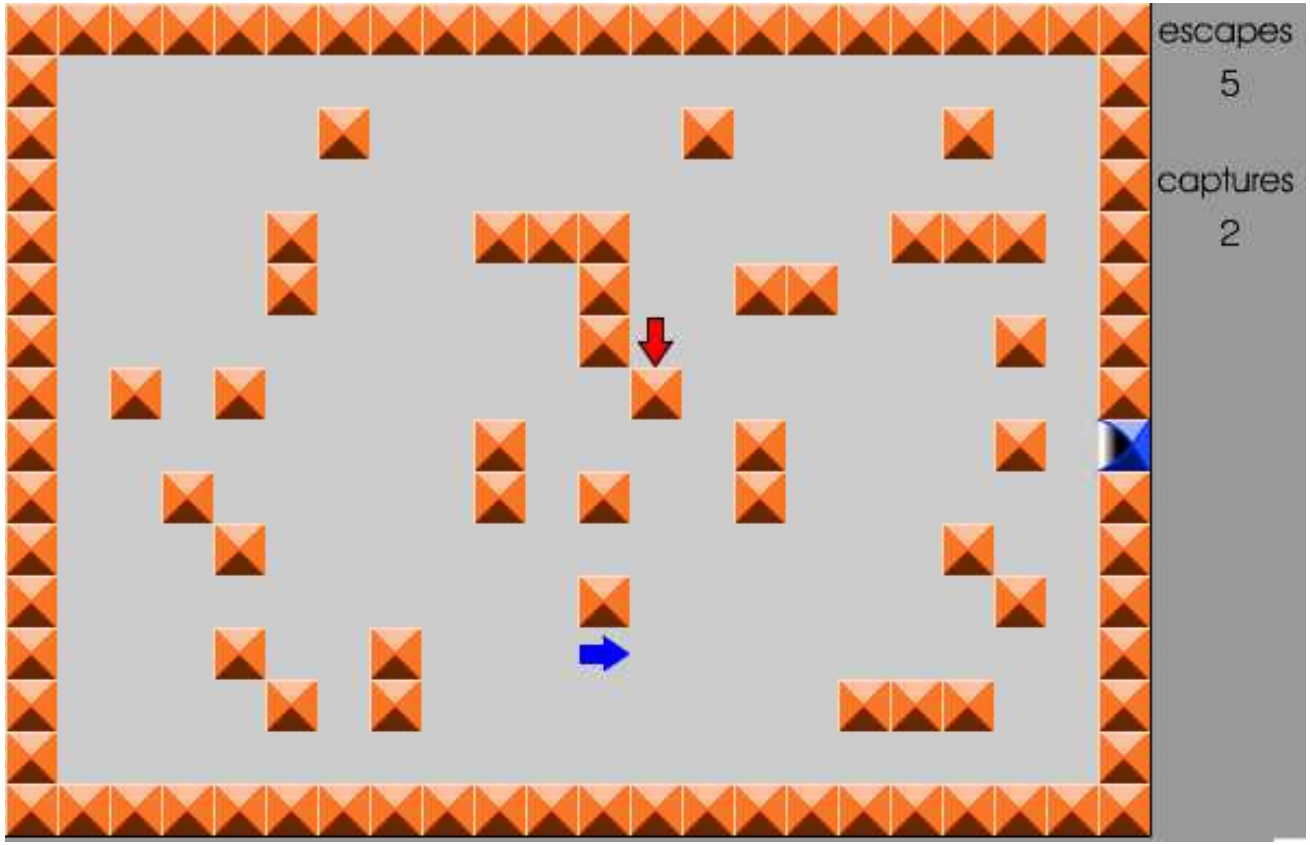
Elements of Logic Game Programming

- Typical structures:
  - Regularly located repetitions of objects
  - Random choice of object state (to create a start situation)
- Design of animation and graphics:
  - Simple and unobtrusive: The user wants to think about a strategy
- Game logic design:
  - Keep problem solvable (sometimes extremely difficult for the author)
  - Avoid too simple solution strategies

- Using Flash or comparable authoring tools:
  - Very effective tool for producing simple games in short time
  - Known Flash patterns are useful:
    - » “Start frame code” (in example: initialization)
    - » “Graphical response” (in example: toggling the mole state)
    - » “Names and numbers” (in example: determining the instance names)

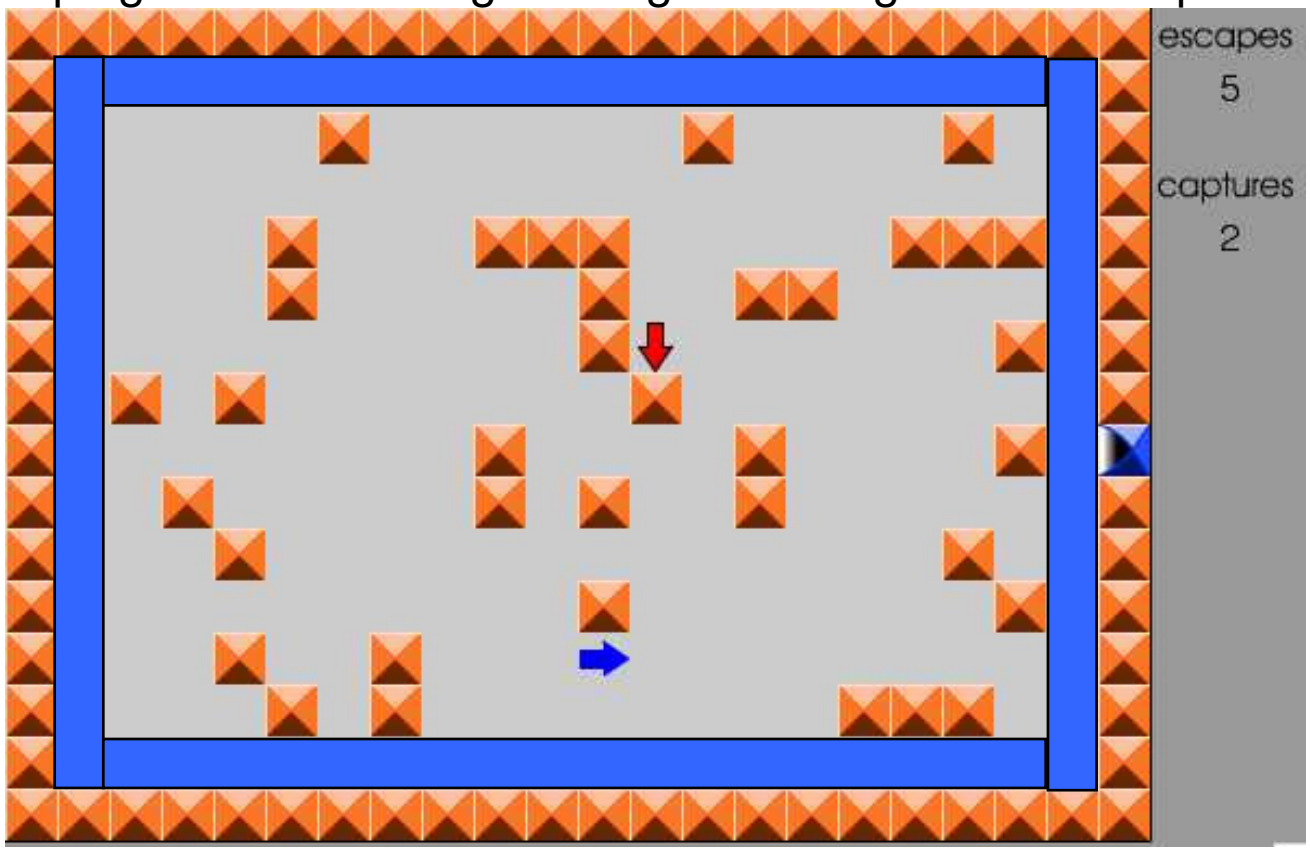
Example 2: Monstachase

- Objective of the game: Try to reach the exit before the monster reaches you. The monster, however, automatically moves towards you.
- Many details of realization very similar to example 1; below just a few additional techniques are discussed



How to Avoid Unsolvable Situations

- Random creation of obstacles may lead to a situation which is unfair to the player (no path to exit)
- Look for simple solutions:
  - E.g. keeping a corridor along the edges of the game-board open



Easy Correlation Between Model and Graphics

- The game-board is represented as a data structure *(model)*
  - 2-dimensional array of numbers representing states
- The number stored for a position represents the state of the position and its visual rendering:
  - 0: empty floor; 1: obstacle; 2: exit
- Appropriate rendering is achieved by the “Graphical Response” pattern:

```
for i, j … {
  s.gotoAndStop(gameboard[i][j]+1)
}
```

Game Levels

- Experienced players may want to play against more difficult challenges
- In logic-based games, adjusting the level can be quite easy

- Example:

```
for i, j …{
   gameboard[i][j] = (Math.random() < .5/diff);
 }
```

  - *diff* = 1 : Chance for game-board position to be solid is 1:2
  - Increasing values of *diff*: Chance for board position to be solid decreases

- Possible improvement of game:
  - When exit is reached, increase *diff* by 1

3 Introduction to Computer Game Programming

3.1 Logic-Based Games

3.2 Graphical Design of Game Characters

- Optimizing Vector Graphics
- Principles of Animation
- Creating a Story and a Hero

3.3 Simulation-Based Games

3.4 Interaction and Sound

Literature:

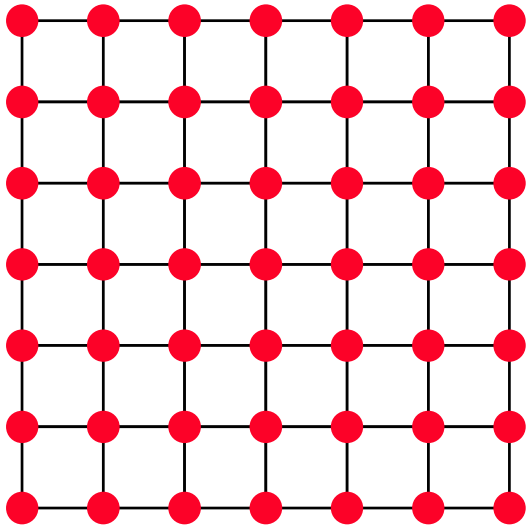
K. Besley et al.: Flash MX 2004 Games Most Wanted, Apress/Friends of ED 2004

Section 3.2 based on book chapter 2 by ***Brad Ferguson***

Source code for all examples at www.friendsofed.com

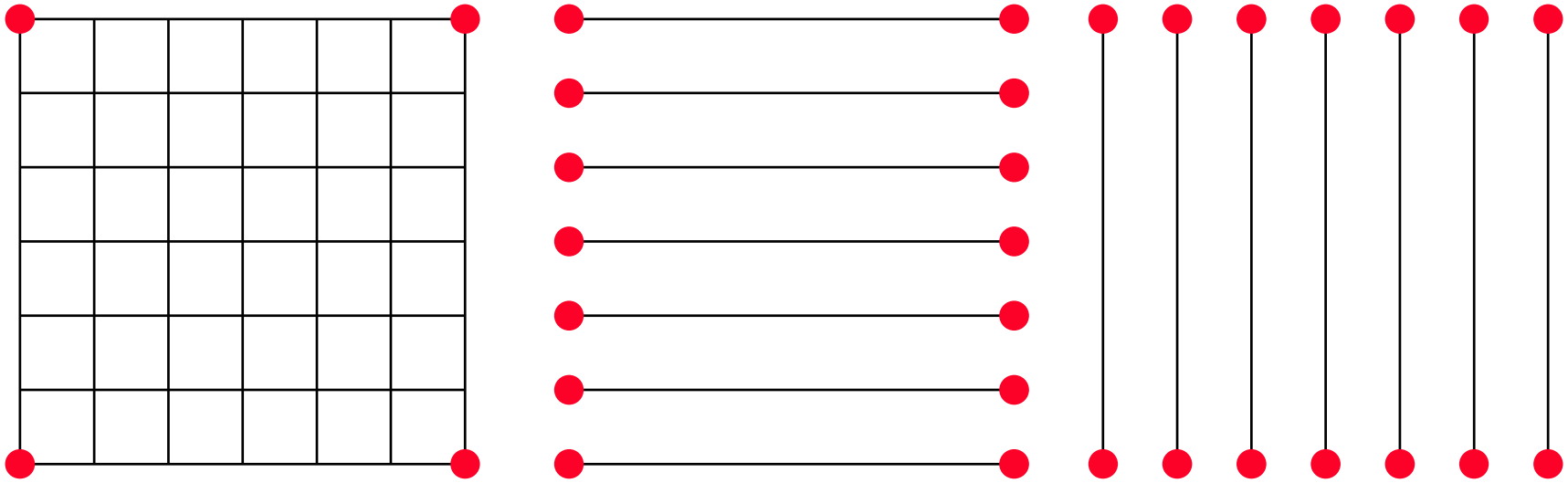
Complexity of Polygon Drawings

- Normal drawings in vector graphics programs (like Flash)
  - Every line has a vector point on each end
  - Every time a line makes a sharp bend, at least one new vector point is needed
  - Every time two lines intersect, yet another vector point is created



49 vector points

Optimized Vector Drawings



4+7*4 = 32 vector points

Crosshair principle: avoid intersections

Flash:

Use separate layers to draw lines which intersect.
Keep each layer free of intersections.

Optimizing a Comic Character

3 Introduction to Computer Game Programming

3.1 Logic-Based Games

3.2 Graphical Design of Game Characters

- Optimizing Vector Graphics
- Principles of Animation
- Creating a Story and a Hero

3.3 Simulation-Based Games

3.4 Interaction and Sound

Literature:

K. Besley et al.: Flash MX 2004 Games Most Wanted, Apress/Friends of ED 2004

Section 3.2 based on book chapter 2 by ***Brad Ferguson***

Source code for all examples at www.friendsofed.com

Principles of (2D-)Animation

- Keyframes
  - … well known or Flash users
- Squash and Stretch
  - Shape of subject reacts to speed and force of movement
- Timing
  - Timing of movement: Gives animation a sense of weight and gravity
- Anticipation, Action, Reaction, Overlapping Action
  - Anticipation: Build up energy before a movement
  - Reaction: Don’t simply stop, but show the process of stopping
  - Overlapping: Hierarchy of connected objects moves in a complex way
- Arcs
  - Every object follows a smooth arc of movement

Animating a Bouncing Ball

- When the ball is going up, it is fighting the force of gravity and will therefore be slower than when it falls.
- On rise and fall, the ball is stretched to give the illusion it is traveling quickly. This effect should be more extreme on the fall.
- At the top of movement, the ball has a certain hang time.
- As soon as the ball hits the ground (and not before), it gets squashed horizontally.
- A shadow animation increases the optical illusion.
- Please note: These are exaggerations for the sake of a stronger illusion.