Základy počítačové grafiky

Základní principy 2D grafických API

Michal Španěl

Ústav počítačové grafiky a multimédií
Fakulta informačních technologií
Vysoké učení technické v Brně

Brno 2017

Cíl přednášky

Seznámit se s principy 2D grafických API a základy tvorby 2D grafických aplikací.

Jak využít znalosti jednotlivých principů 2D grafiky a ze střípků poskládat aplikaci?

Knihovny pro 2D grafiku

- Knihovny výrobců OS pro různé platformy (Windows, Mac OS X, Android, Windows Phone, atd.)
- Multiplatformní knihovny, které poskytují další abstrakci...
- Knihovny pro webové aplikace (Javascript, atd.)

Příklady

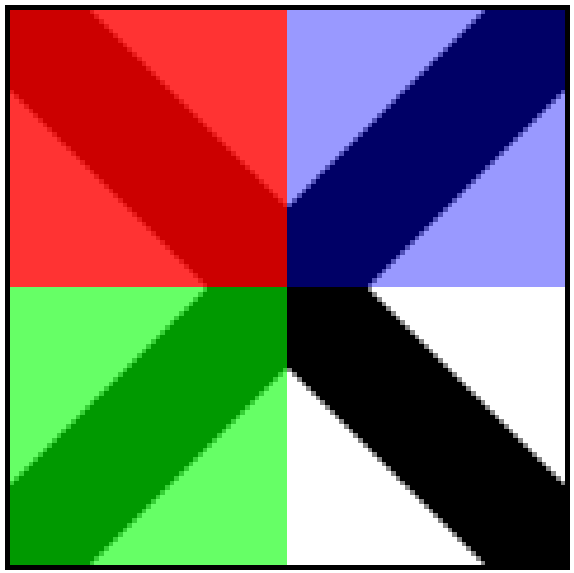
- Cairo (GTK+)
- SDL (znáte z projektu...)
- Skia (Google, použito v Chrome)
- Direct 2D (Windows)
- Quartz 2D (Mac)
- Java 2D

Cairo (http://cairographics.org/)

- Multiplatformní open source knihovna (Windows, Mac OS X, Android, Windows Phone, ...)
- Používá se v GTK+ a Gecko (renderovací jádro pro webové prohlížeče, např. Mozilla)
- Podporuje výstup do obrázku, PDF i SVG
- Pouze kreslení, neřeší interakci s uživatelem...
- Lze snadno propojit s SDL knihovnou...

Příklad použití Cairo

```
cairo_set_source_rgb (cr, 0, 0, 0);
cairo_move_to (cr, 0, 0);
cairo_line_to (cr, 1, 1);
cairo_move_to (cr, 1, 0);
cairo_line_to (cr, 0, 1);
cairo_set_line_width (cr, 0.2);
cairo_stroke (cr);

cairo_rectangle (cr, 0, 0, 0.5, 0.5);
cairo_set_source_rgba (cr, 1, 0, 0, 0.80);
cairo_fill (cr);

cairo_rectangle (cr, 0, 0.5, 0.5, 0.5);
cairo_set_source_rgba (cr, 0, 1, 0, 0.60);
cairo_fill (cr);

cairo_rectangle (cr, 0.5, 0, 0.5, 0.5);
cairo_set_source_rgba (cr, 0, 0, 1, 0.40);
cairo_fill (cr);
```

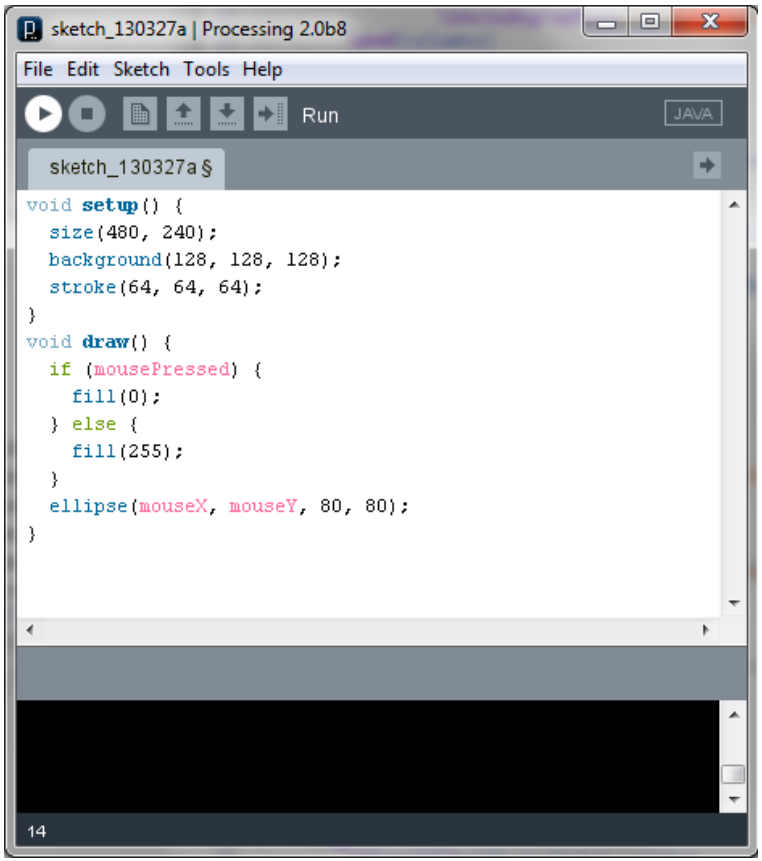


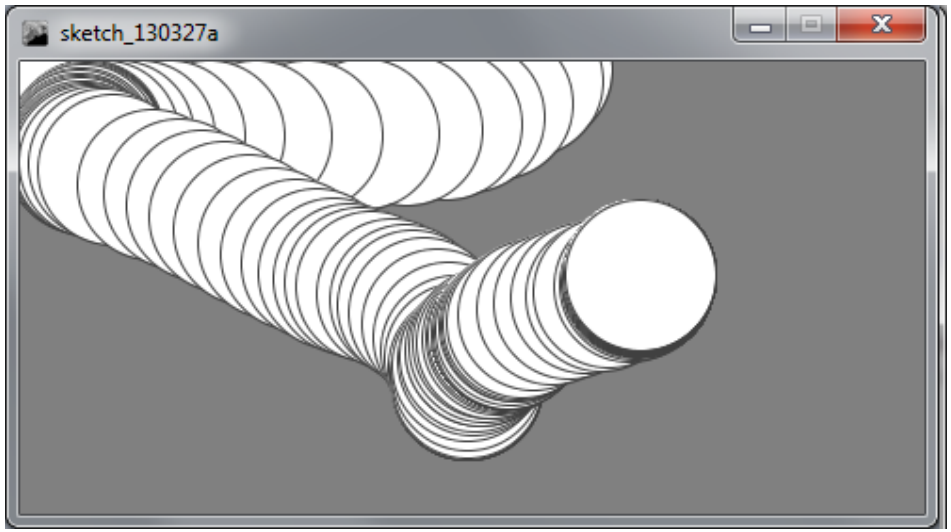
Processing (http://www.processing.org/)

- Jednoduchý programovací jazyk a prostředí (open source, Java)
- 2D a 3D grafika, interaktivní aplikace, animace,. . .
- Rychlé prototypování řešení
- Aplikaci lze „exportovat“ pro spuštění na Windows, Linux, apod.
- Ale i do Javascriptu pro použití na webu
- Budeme používat v přednášce pro ilustrační příklady

Processing - základní prostředí

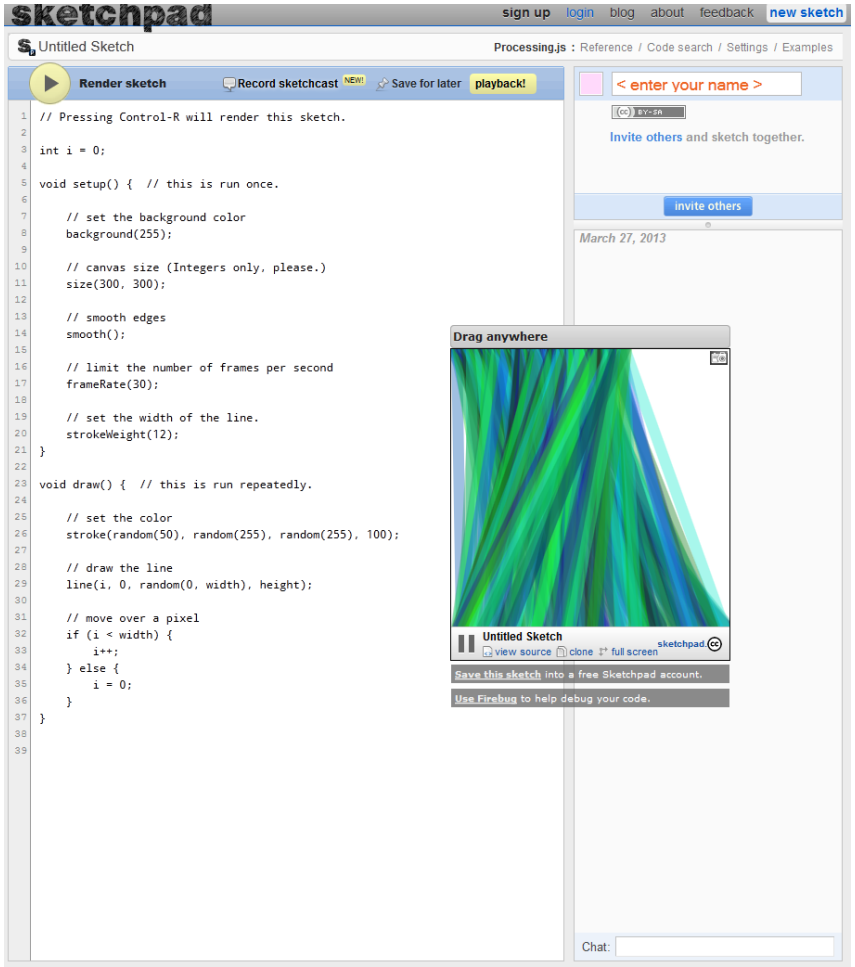
- Stahujte z http://www.processing.org/download/
- Rozbalte ZIP archiv
- A spusťte *processing.exe*





Processing - Studio Sketchpad

- Webové prostředí pro Processing…
- http://sketchpad.cc/



Processing, pokr.

Ukázky

- http://studio.sketchpad.cc/sp/padlist/all-portfolio-sketches
- http://www.processing.org/exhibition/

Dokumentace

- http://www.processing.org/learning/
- http://www.processing.org/reference/

Cíl přednášky

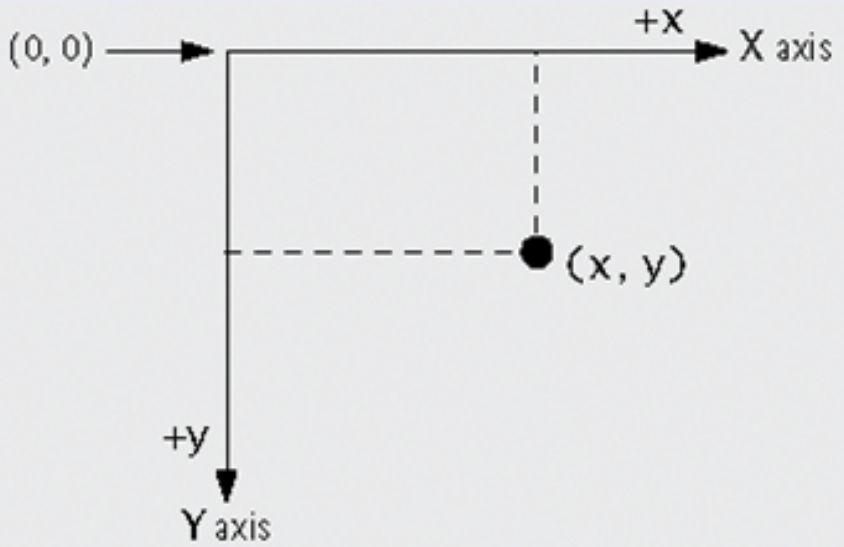
Seznámit se s obecnějšími principy 2D grafických API, které se v různé podobě vyskytují ve všech knihovnách.

Lokalní souřadný systém

- 2D souřadný systém používaný při kreslení “v rámci okna”
- Orientace a pozice počátku $(0, 0)$ dána konkrétním API
- Většinou v jednotkách pixelů (floating-point hodnoty)

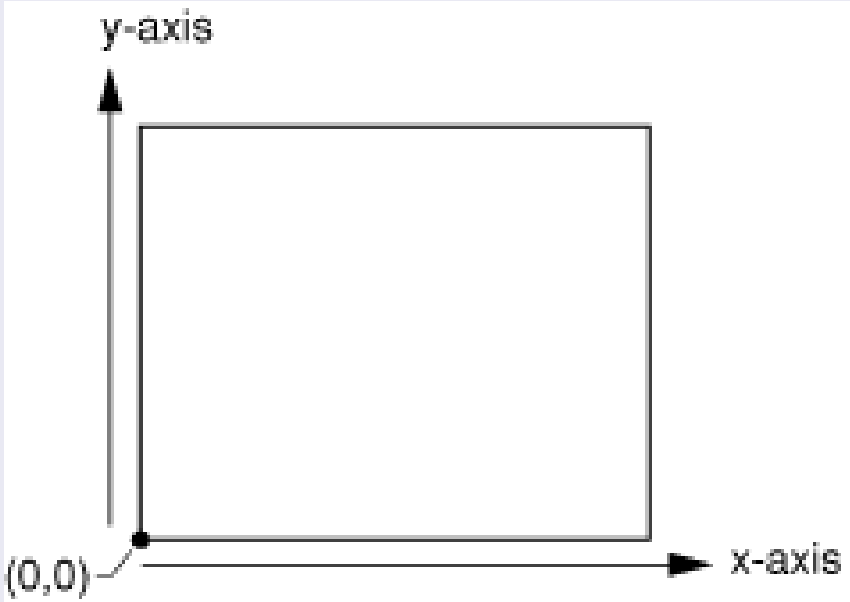
Počátek nahoře

- např. Cairo, SDL, Java 2D a Processing



Počátek dole

- např. Quartz 2D

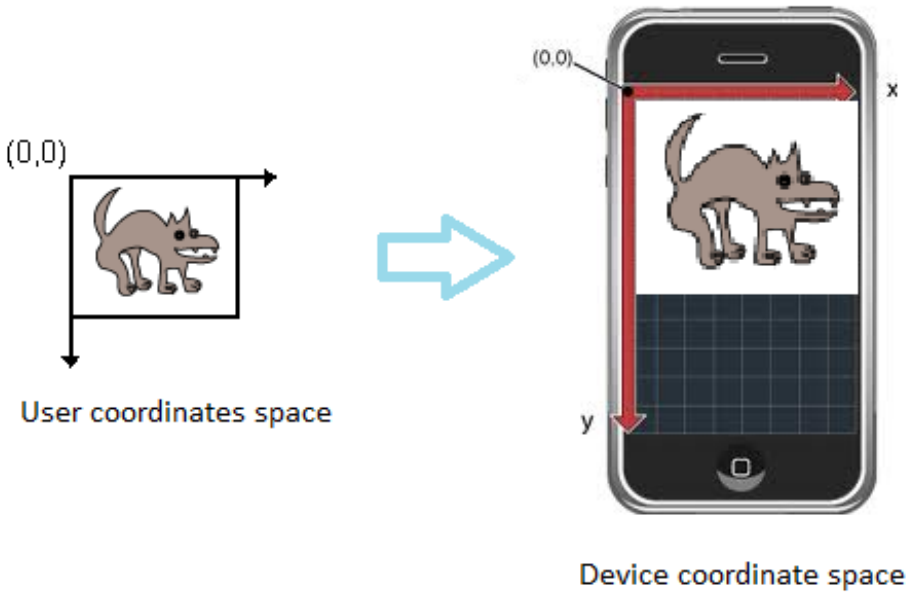


Souřadný systém nezávislý na zařízení

- Různá zařízení (monitor, tiskárna, telefon) mají různé grafické možnosti (rozlišení, apod.).
- Většina API definuje:
  - souřadný systém zařízení (*device coordinate system*)
  - uživatelský souřadný systém (*user coordinate space*)

Přepočet mezi souřadnicemi

- Objekty se definují v uživatelském souř. systému.
- Před vykreslením se aplikuje transformační matice.
- Typicky změna měřítka.

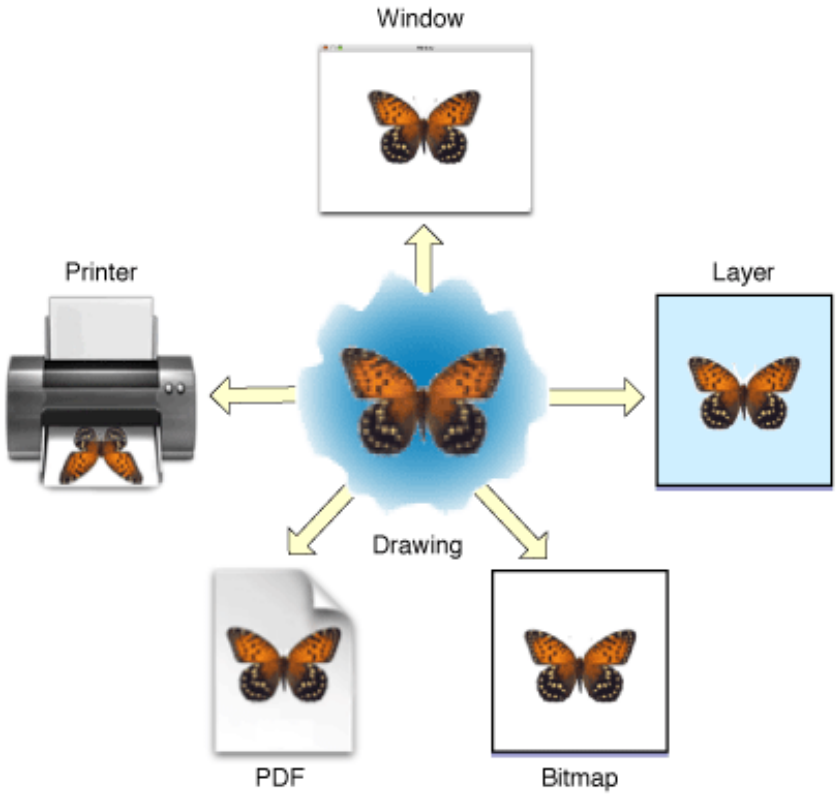


Co je to grafický kontext?

- „Cíl kreslení“ (též. graphics context, surface, render target, ...)

Grafický kontext

- Datová struktura, která drží specifické informace potřebné pro kreslení na různá výstupní zařízení (display, bitmapa, PDF soubor, atd.).



Vytvoření kontextu v knihovně Cairo

```
cairo_surface_t *surface;
cairo_t *cr;

surface = cairo_image_surface_create( CAIRO_FORMAT_ARGB32,
                                      120, 120 );
cr = cairo_create( surface );

cairo_set_line_width( cr, 30.0 );
cairo_set_source_rgb( cr, 1, 0.2, 0.2 );
cairo_set_line_cap( cr, CAIRO_LINE_CAP_BUTT );
cairo_move_to( cr, 64.0, 50.0 );
cairo_line_to( cr, 64.0, 200.0 );
cairo_stroke( cr );
```

Co grafický kontext obsahuje?

- Parametry výstupního zařízení (formát obrázku, atd.)
- Šířka a výška kreslící plochy (řeší i ořezávání)
- Transformace výstupu (device-independent kreslení)

„Statefull“ grafický kontext (Cairo, SDL, Quartz 2D, . . .)

- Stavové proměnné pro různá nastavení kreslení
- Kreslící barva
- Tloušťka čáry
- atd.

„Stateless“ grafický kontext (Skia, Direct 2D, . . .)

- Nastavení kreslení není součást kontextu
- Samostatné struktury. . .

„Stateless“ grafický kontext v Direct2D

Vytvoření štětce

```
// Create a gray brush.
m_pRenderTarget->CreateSolidColorBrush(
    D2D1::ColorF(D2D1::ColorF::LightSlateGray),
    &m_pLightSlateGrayBrush
    );
```

Kreslení...

```
m_pRenderTarget->DrawLine(
    D2D1::Point2F(0.0f, 0.0f),
    D2D1::Point2F(0.0f, 50.0f),
    m_pLightSlateGrayBrush,
    0.5f
    );
```

Co jsou to události?

- Umožňují interakci s uživatelem i reakci na změny ve stavu aplikace, operačního systému, apod.

Události pocházejí od

- Klávesnice, myši, joysticku, ...
- OS – změna velikosti okna, překreslení okna, ...
- Vlastní události (změna ve stavu aplikace, hotovo načtení dat, ...)
- Časovač (např. animace)

Realizace událostí

- Jak se mechanismus událostí prakticky realizuje?

Zasílání zpráv (WinAPI)

```
LRESULT CALLBACK WndProc(...)
{
  switch(Msg)
  {
  case WM_CREATE: break;
  case WM_SIZE: break;
  case WM_DESTROY:
    PostQuitMessage(WM_QUIT);
    break;
  default:
    return DefWindowProc(...);
  }
  return 0;
}
LRESULT SendMessage(...);
BOOL GetMessage(...);
```

Callback funkce (GLUT, časovač v SDL)

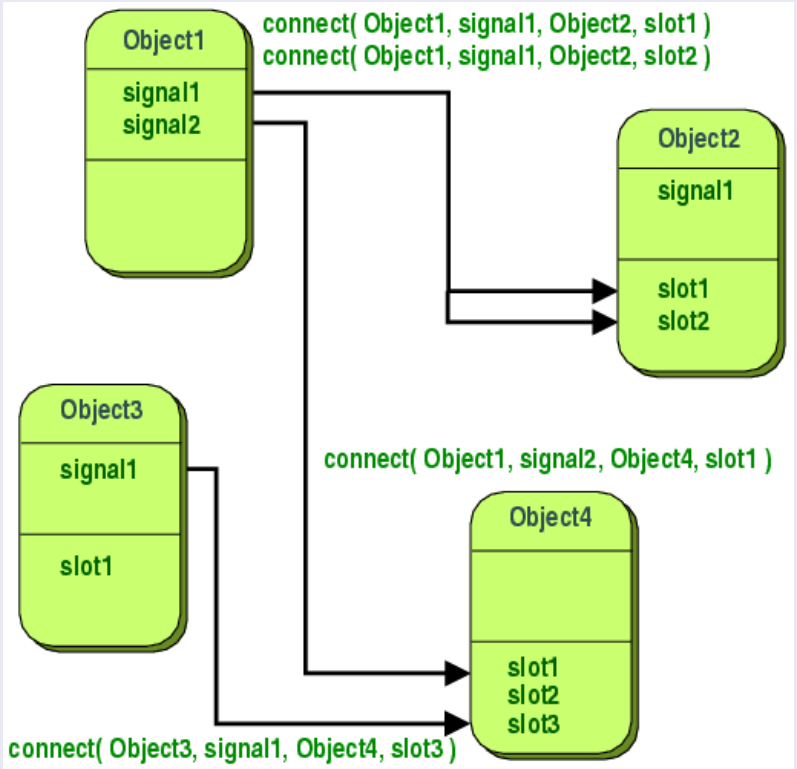
- Callback je ukazatel na fci

```
Uint32 onTimer(...)
{
    SDL_Event event;
    event.type = SDL_USEREVENT;
    ...
    SDL_PushEvent(&event);

    return interval;
}

// registrace callback fce
SDL_AddTimer(80, onTimer, NULL);
```

Realizace událostí, pokr.

Signály a sloty (Qt)

- Objektové jazyky (C++, . . .)
- Lepší typová kontrola



Smyčka událostí/zpráv

- Typická součást kódu GUI knihoven pro zpracování událostí

Smyčka událostí v SDL

```
SDL_Event event;

while( SDL_PollEvent(&event) )
{
  switch (event.type) {
    case SDL_MOUSEMOTION:
      printf("Mouse moved by %d,%d to (%d,%d)\n",
             event.motion.xrel, event.motion.yrel,
             event.motion.x, event.motion.y);
      break;
    case SDL_QUIT:
      exit(0);
    }
}
```

Kdy kreslit?

- Pokud reagujeme na vstup uživatele
- Je-li třeba překreslit okno (změna velikosti, maximalizace, apod.)
- Něco ve scéně se změnilo…

Fce pro vykreslení scény (viz projekt v SDL)

```
SDL_Surface * screen = NULL;

// funkce zajistujici prekresleni obsahu okna
void draw(void)
{
  // nejake to kresleni
  ...

  // vymena zobrazovaneho a zapisovaneho bufferu
  SDL_Flip(screen);
}
```

Kdy kreslit, pokr.

Smyčka zpráv (viz projekt v SDL)

```
int main(int argc, char *argv[])
{
  SDL_Event event;

  if( SDL_Init(SDL_INIT_VIDEO) == -1 ) exit(-1);
  screen = SDL_SetVideoMode(DEFAULT_WIDTH, DEFAULT_HEIGHT, ...);

  while( !quit ) {
    while( SDL_PollEvent(&event) ) {
      switch( event.type )
      {
        case SDL_VIDEORESIZE: // zmena velikosti okna
          onResize(&event.resize);
          break;
        case SDL_QUIT: // SDL_QUIT event
          quit = 1;
          break;
        ...
      }
    }

    // vykresleni po jakekoliv udalosti
    draw();
  }
```

„OnPaint“ (Passive Rendering)

- Událost/zpráva generovaná GUI když je třeba překreslit okno.
- WM_PAINT, QPaintEvent, wxPaintEvent, ...

Zpráva WM_PAINT v D2D

```
LRESULT MainWindow::HandleMessage(...)
{
  switch (uMsg)
  {
    case WM_PAINT:
      OnPaint();
      return 0;
    ...
  }
  return DefWindowProc(m_hwnd, uMsg, wParam, lParam);
}
```

Kreslení v Processing (http://sketchpad.cc/)

- Automaticky volané vestavěné funkce
- Globalní proměnné (pozice myši, velikost okna, atd.)

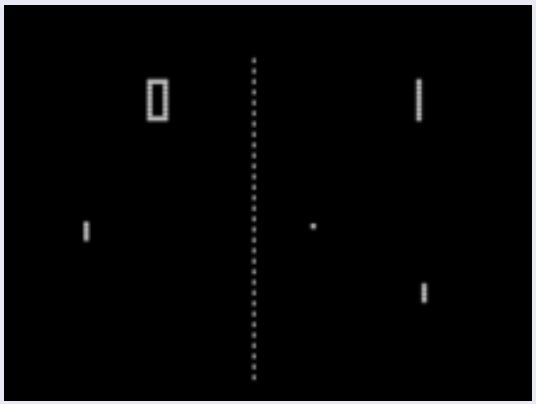
Reakce na myš

```
void setup() {
  size(480, 240);
  background(128, 128, 128);
  stroke(64, 64, 64);
}
void draw() {
  if (mousePressed) {
    fill(0);
  } else {
    fill(255);
  }
  ellipse(mouseX, mouseY, 80,
      80);
}
```

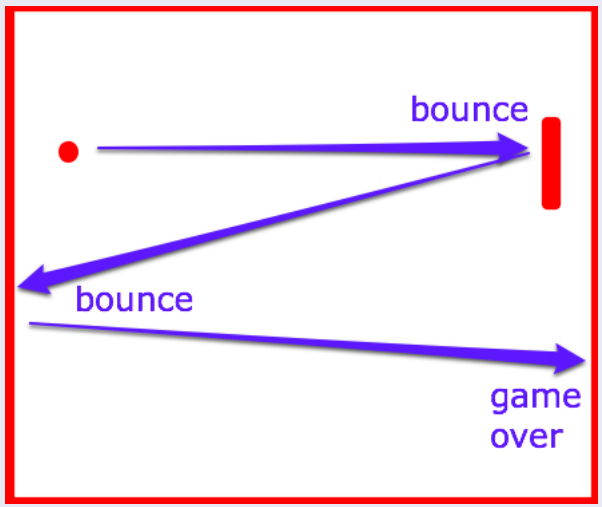
- setup() . . . zavolá se jednou (inicializace)
- draw() . . . volá se opakovaně (kreslení)
- mousePressed . . . stav myši
- mouseX, mouseY . . . pozice kurzoru v okně

Hra „Pong“

Allan Alcorn (Atari), 1976



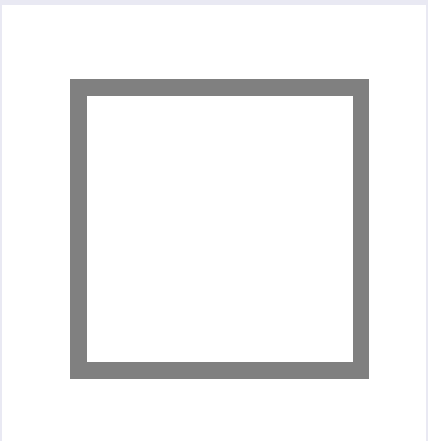
Naše verze pro jednoho hráče...



Stroke vs. Fill

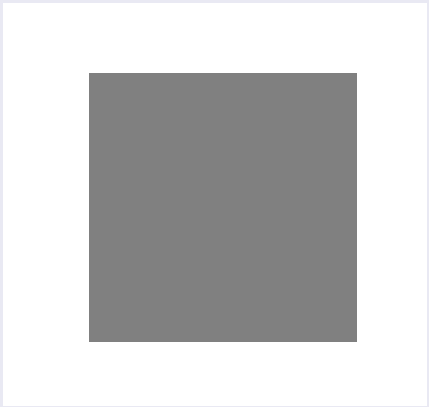
Tah/Stroke

- Čáry a obrysy objektů:
  - kreslící barva,
  - šířka čáry,
  - styl čáry (plná, čárkovaná, . . .),
  - typ štětce/pera, apod.



Výplň/Fill

- Vnitřní část objektu:
  - barva,
  - gradient,
  - vzory, apod.



Různé přístupy k implementaci...

Processing

```
size(100, 100);
background(255);
...
strokeWeight(3);
// barva
stroke(128);
// zadani a kresleni tvaru
rect(25, 25, 50, 50);
...
// bez hranice
noStroke();
// barva vyplne
fill(128);
// kresleni
rect(25, 25, 50, 50);
```

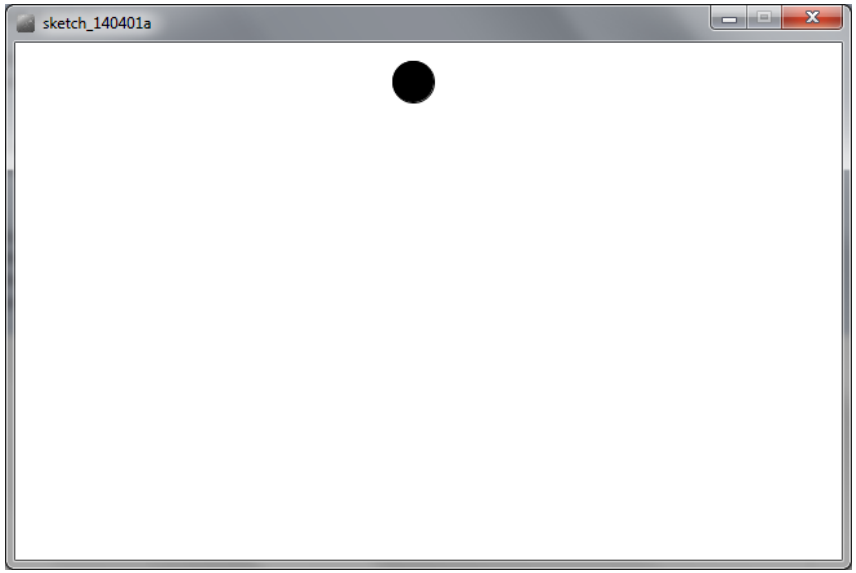
Cairo

```
// barva
cairo_set_source_rgb(cr, 0.5, 0.5, 0.5);
// zadani tvaru
cairo_rectangle(cr, 25, 25, 50, 50);
...
cairo_set_line_width (cr, 3.0);
// kresleni obrysu tvaru
cairo_stroke(cr);
...
// kresleni vyplneneho tvaru
cairo_fill(cr);
```

Hra „Pong“ (1)

Letící míček v ose X

```
float posX = 30;
float posY = 30;
float ballR = 16;

void ball()
{
  fill(0);
  ellipse(posX, posY, 2 * ballR, 2 * ballR);
}

void setup() {
  size(640, 400);
}

void draw() {
  background(255, 255, 255);
  ball();
  posX = posX + 1;
}
```



Hra „Pong“ (2)

Odraz od stěny

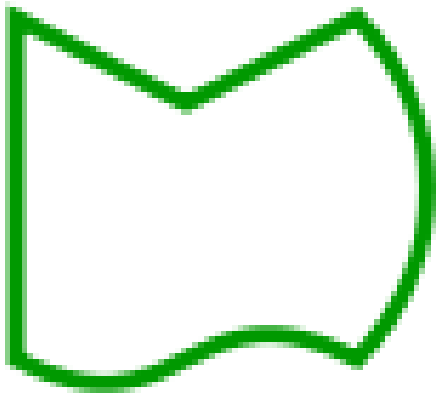
```
float posX = 30;
float posY = 30;
float ballR = 16;
float dX = 1;

void ball() {
  fill(0);
  ellipse(posX, posY, 2 * ballR, 2 * ballR);
}

void setup() {
  size(640, 400);
}

void draw() {
  background(255, 255, 255);
  ball();
  posX = posX + dX;
  if( posX > width || posX < 0 ) {
    dX = -dX;
  }
}
```

Co dál?

- Podobně vyřešit osu Y
- Ošetřit správný okamžik odrazu
- Náhodná inicializace

Path

- Definice složitějších tvarů - polygony, křivky, atd.

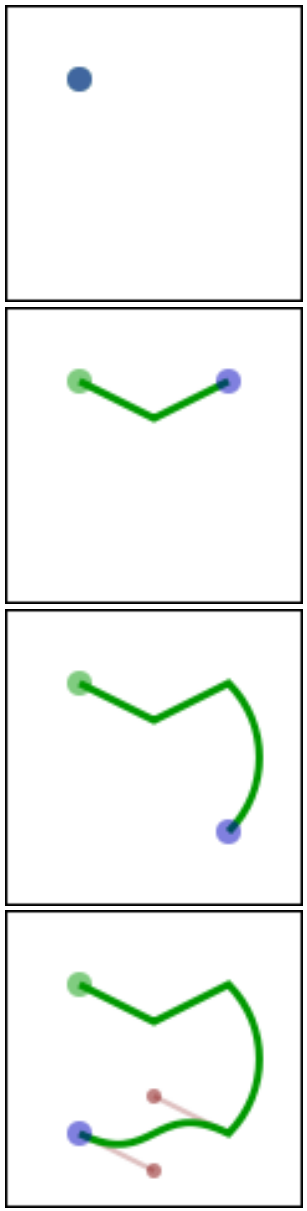


Cairo

- „Aktivní“ cesta je součástí kontextu
- Zadávání stylem propojování bodů

```
cairo_stroke(); // vykreslí hranici
cairo_fill(); // vyplní tvar
```

Příklad v Cairo (http://cairographics.org/tutorial/)

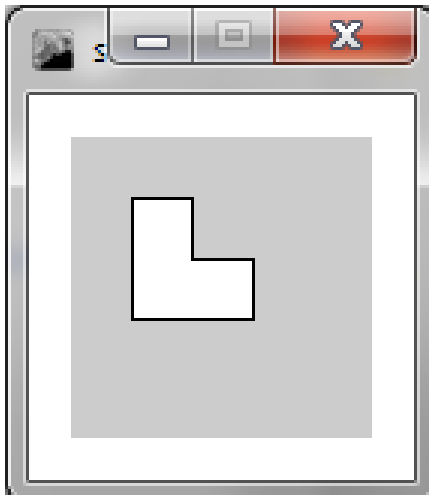


```
cairo_move_to (cr, 0.25, 0.25);

// propojovani bodu - cary
cairo_line_to (cr, 0.5, 0.375);
cairo_rel_line_to (cr, 0.25,
    -0.125);

// oblouk
cairo_arc (cr, 0.5, 0.5, 0.25 *
    sqrt(2), -0.25 * M_PI, 0.25
     * M_PI);

// Bezierova kubika
cairo_rel_curve_to (cr, -0.25,
    -0.125, -0.25, 0.125, -0.5,
     0);

// uzavreni tvaru
cairo_close_path (cr);
```

Shapes – alternativa v Processing

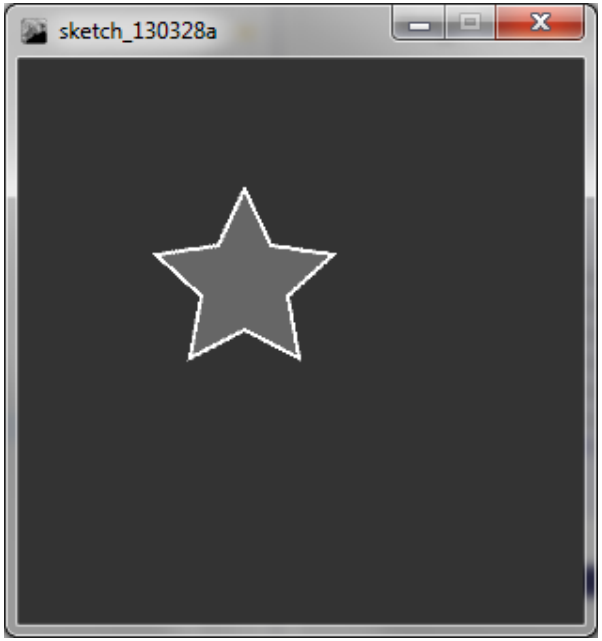
- Definice složitějších tvarů
- beginShape() a endShape()

```
void setup()
{
  size(100, 100);

  beginShape();
  vertex(20, 20);
  vertex(40, 20);
  vertex(40, 40);
  vertex(60, 40);
  vertex(60, 60);
  vertex(20, 60);
  endShape(CLOSE);
}
```



Shapes – objektově

```
PShape star;

void setup() {
  size(300, 300, P2D);
  star = createShape();
  star.beginShape();
  star.fill(102);
  star.stroke(255);
  star.strokeWeight(2);
  star.vertex(0, -50);
  star.vertex(14, -20);
  star.vertex(47, -15);
  star.vertex(23, 7);
  star.vertex(29, 40);
  star.vertex(0, 25);
  star.vertex(-29, 40);
  star.vertex(-23, 7);
  star.vertex(-47, -15);
  star.vertex(-14, -20);
  star.endShape(CLOSE);
}

void draw() {
  background(51);
  translate(mouseX, mouseY);
  shape(star);
}
```



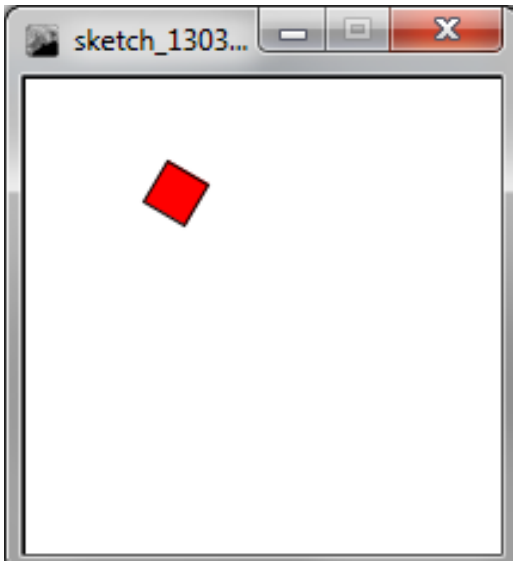
2D transformace

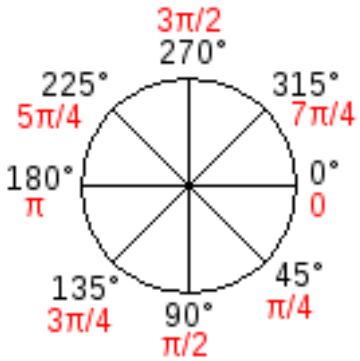
- Afinní transformace
- Zadávají se pomocí fcí typu
  - translate(tx, ty)
  - rotate(rads)
  - scale(sx, sy)
- Akumulují se v „globální transformační matici“
- Nastavené transformace se aplikují při kreslení objektů

```
void setup() {
  size(200, 200);
  background(255);
  noStroke(); // no outline

  // draw the original position
  fill(192);
  rect(20, 20, 40, 40);

  // changing the coordinates
  fill(255, 0, 0, 128);
  rect(20 + 60, 20 + 80, 40, 40);

  // and using transformations
  fill(0, 0, 255, 128);
  translate(60, 80);
  rect(20, 20, 40, 40);
}
```

Na pořadí transformací záleží...

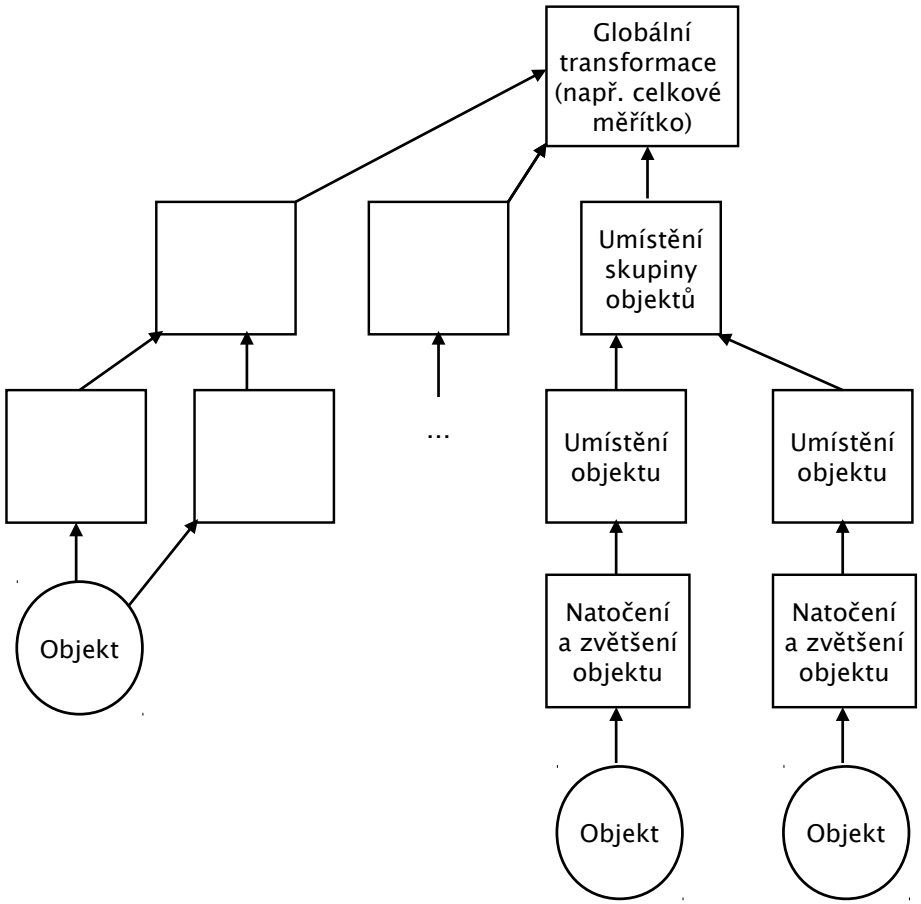
- V jakém pořadí se transformace aplikují?






```
void setup() {
  size(200, 200);
  background(255);
  smooth(); // antialiased edges
  line(0, 0, 200, 0); // draw axes
  line(0, 0, 0, 200);

  fill(255, 0, 0); // red square
  rotate(radians(30));
  translate(70, 0);
  rect(0, 0, 20, 20);
}
```

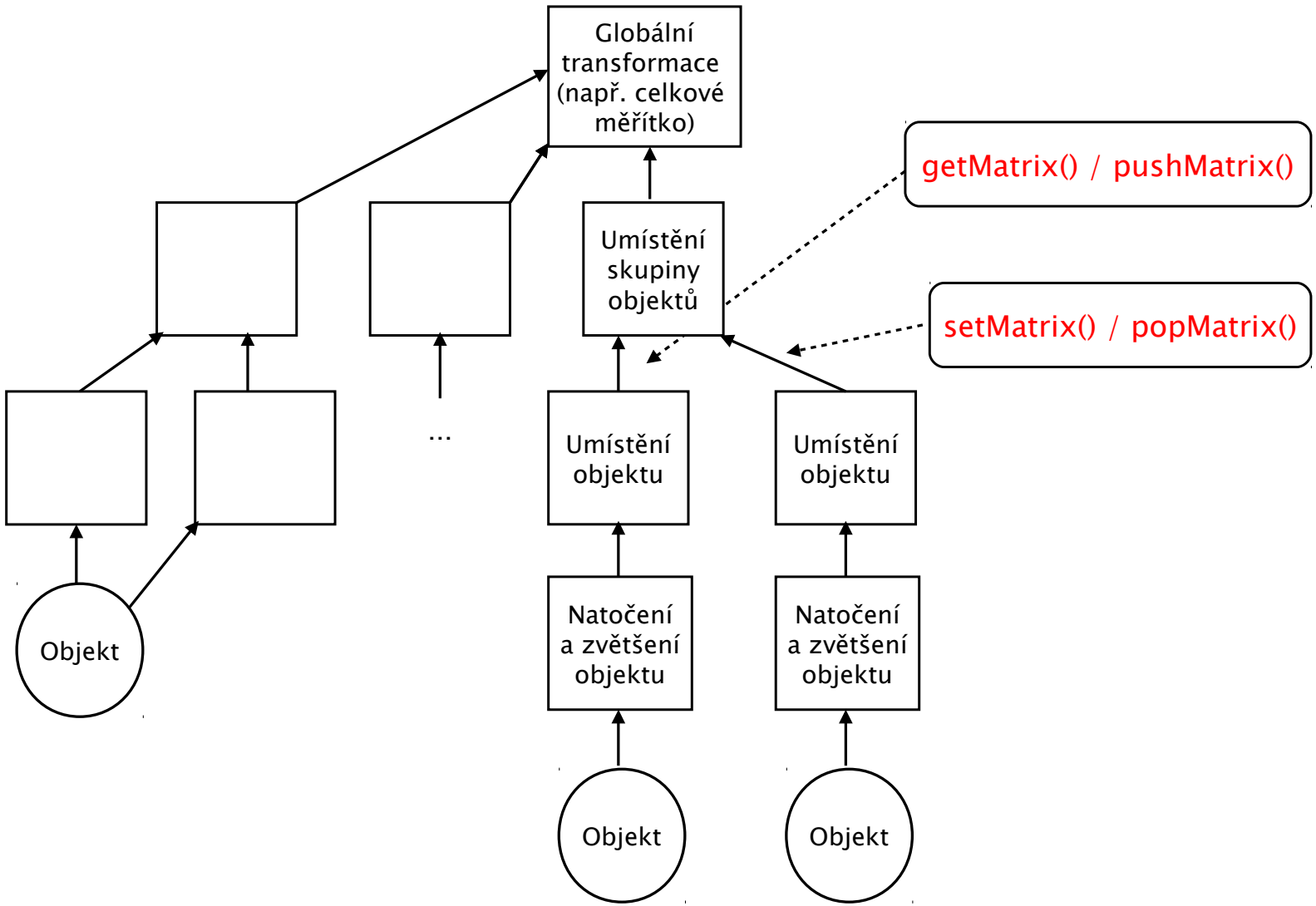
„Strom transformací“



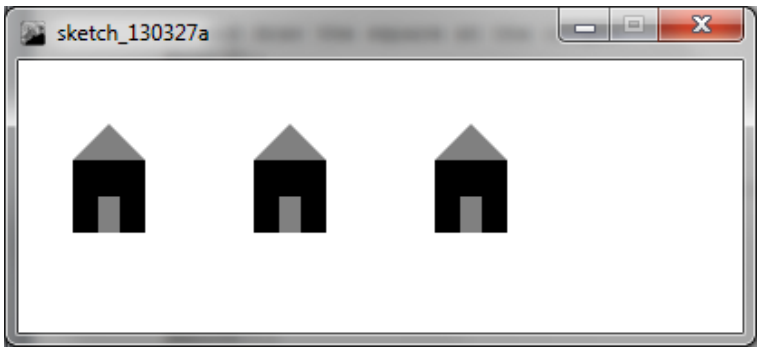
- Hierarchická představa transformací objektů
- Uzly . . . transformace
- Listy . . . kreslení objektu
- Skládáme transformace v uzlech po cestě ke kořenu

- Jak realizovat prakticky?

(get,set)Matrix nebo (push,pop)Matrix



Příklad v Processing

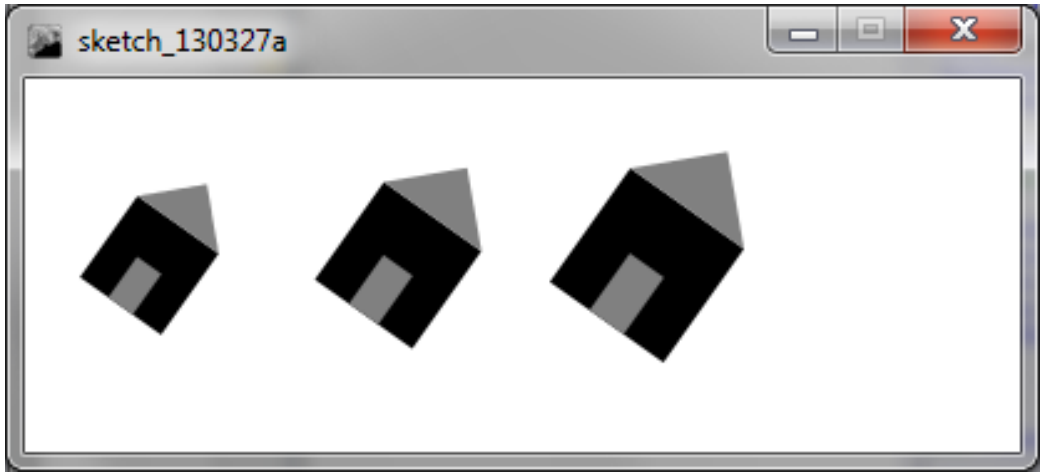



```
void setup() {
  size(400, 150);
  background(255);
  noStroke();

  translate(50, 75);
  for( int i = 0; i < 3; ++i )
  {
    pushMatrix();
    translate(i * 100, 0);
    scale(20);
    house();
    popMatrix();
  }
}

void house() {
  fill(0);
  rect(-1, -1, 2, 2);
  fill(128);
  triangle(-1, -1, 1, -1, 0, -2);
  rect(-0.3, 0, 0.6, 1);
}
```

Vyzkoušejte si...

Upravit příklad tak, aby

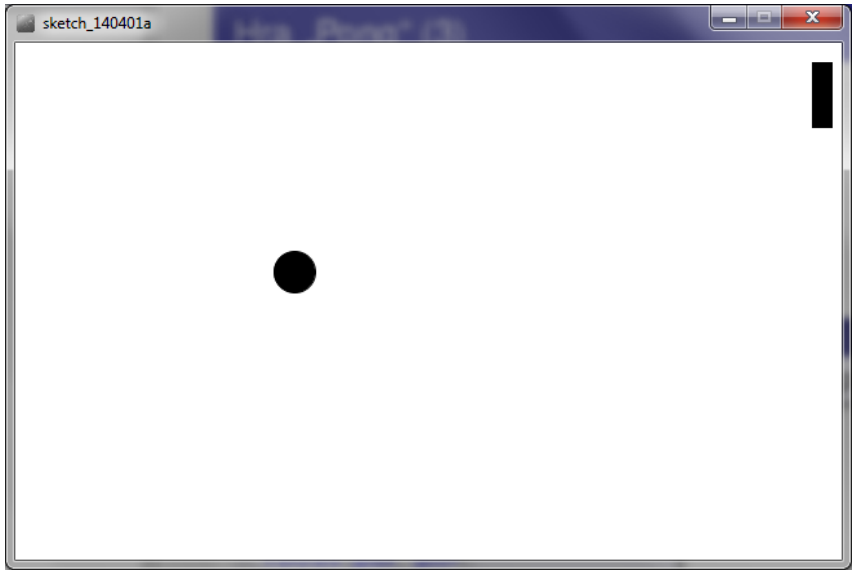
- Domečky byly různě velké
- a otáčely se na místě...



Hra „Pong“ (3)

Přidáme transformace a pálku

```
float pposX, float pposY = 15;
float paddleW = 15; float paddleH = 50;

void ball() {
  fill(0);
  ellipse(0, 0, 2 * ballR, 2 * ballR);
}
void paddle() {
  fill(0);
  rect(0, 0, paddleW, paddleH);
}
void setup() {
  size(640, 400);
  pposX = width - 1.5 * paddleW;
}
void draw() {
  background(255, 255, 255);

  pushMatrix();
  translate(posX, posX);
  ball();
  popMatrix();

  translate(pposX, pposY);
  paddle();
  ...
```



Co dál?

- Jak na pohyb pálkou...

Hra „Pong“ (4)

Pohyb pálkou

```
void draw() {
  ...
  if (keyPressed) {
    if (key == CODED) {
      if(keyCode == UP) {
        if (pposY >= 0) {
          pposY = pposY - 5;
        }
      }
      if(keyCode == DOWN) {
        if(pposY <= height - paddleH) {
          pposY = pposY + 5;
        }
      }
    }
  }
}
```

Co dál?

- Kolize...
- Konec hry...

Hra „Pong“ (5)

Kolize a konec hry

```
void draw() {
  ...
  if( collision() ) {
    dX = -dX;
  }
  if( posX + ballR > width ) {
    fill(255, 0, 0, 100);
    rect(0, 0, width, height);
    noLoop();
  }
  ...
}
boolean collision() {
  if (posX + ballR >= pposX) {
    if ((posY >= pposY) && (posY <= pposY +
        paddleH)) {
      return true;
    }
  }
  return false;
}
```

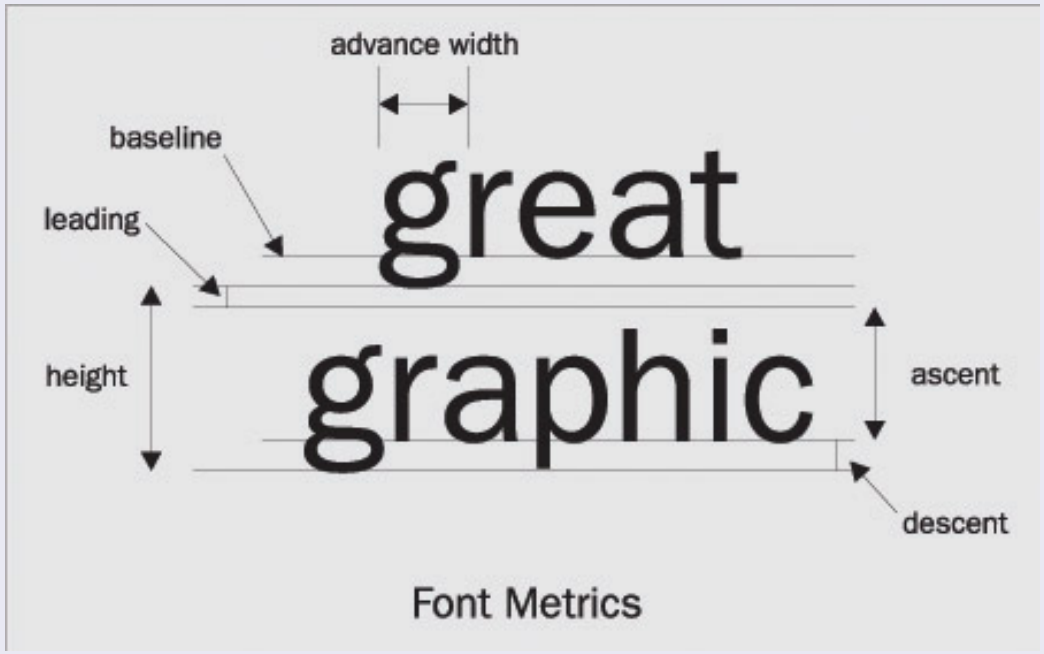
Co dál?

- Výpis skóre...

Použití fontů

- Fonty definovány vektorově
- Text se chápe jako další tvar (cesta)
- Lze kreslit hranici nebo vyplnit
- Výběr fontu
  - Serif, Arial, ...
  - Styl fontu (plain, bold, italic)
  - Velikost fontu
  - atd.

Definice a parametry fontů



Fonty v Processing

```
PFont f;

void setup()
{
  size(200, 200);
  background(255);

  // Arial, 16 point, anti-aliasing on
  f = createFont("Arial", 36, true);

  // font to be used
  textFont(f, 36);

  // font color
  fill(128);

  // draw text
  text("Hello world!", 10, 100);
}
```

Hra „Pong“ (6)

Výpis skóre

```
PFont f;
int score = 0;

void draw() {
  ...
  if( collision() ) {
    dX = -dX;
    score = score + 1;
  }
  ...
  textFont(f, 32);
  fill(150, 0, 0);
  text("Score: " + str(score), 150, 50);
}
```

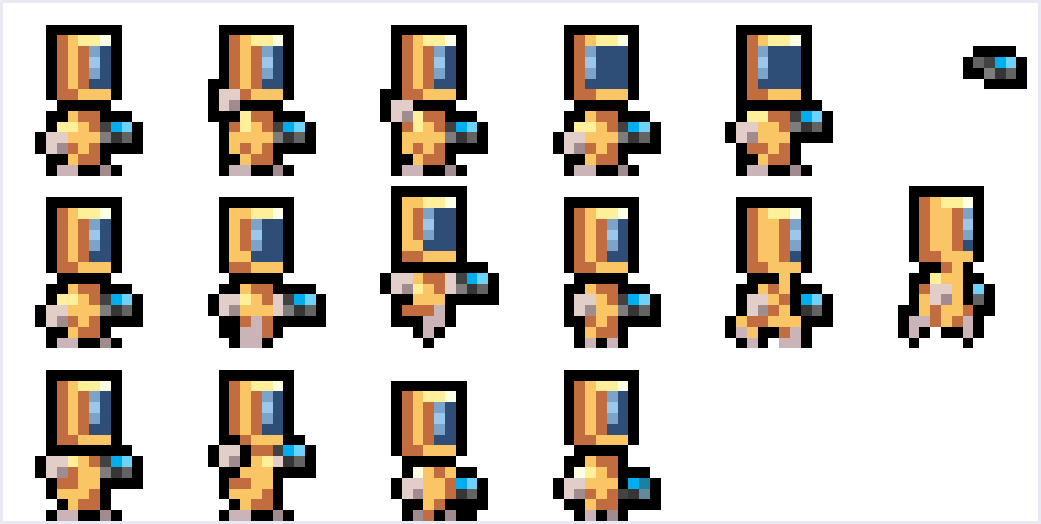
Co dál?

- Lepší grafika...

Co je to Sprite?

- Grafika, založená na bitmapových obrázcích
- Animace pohyblivých objektů
- Typicky 2D hry, apod.

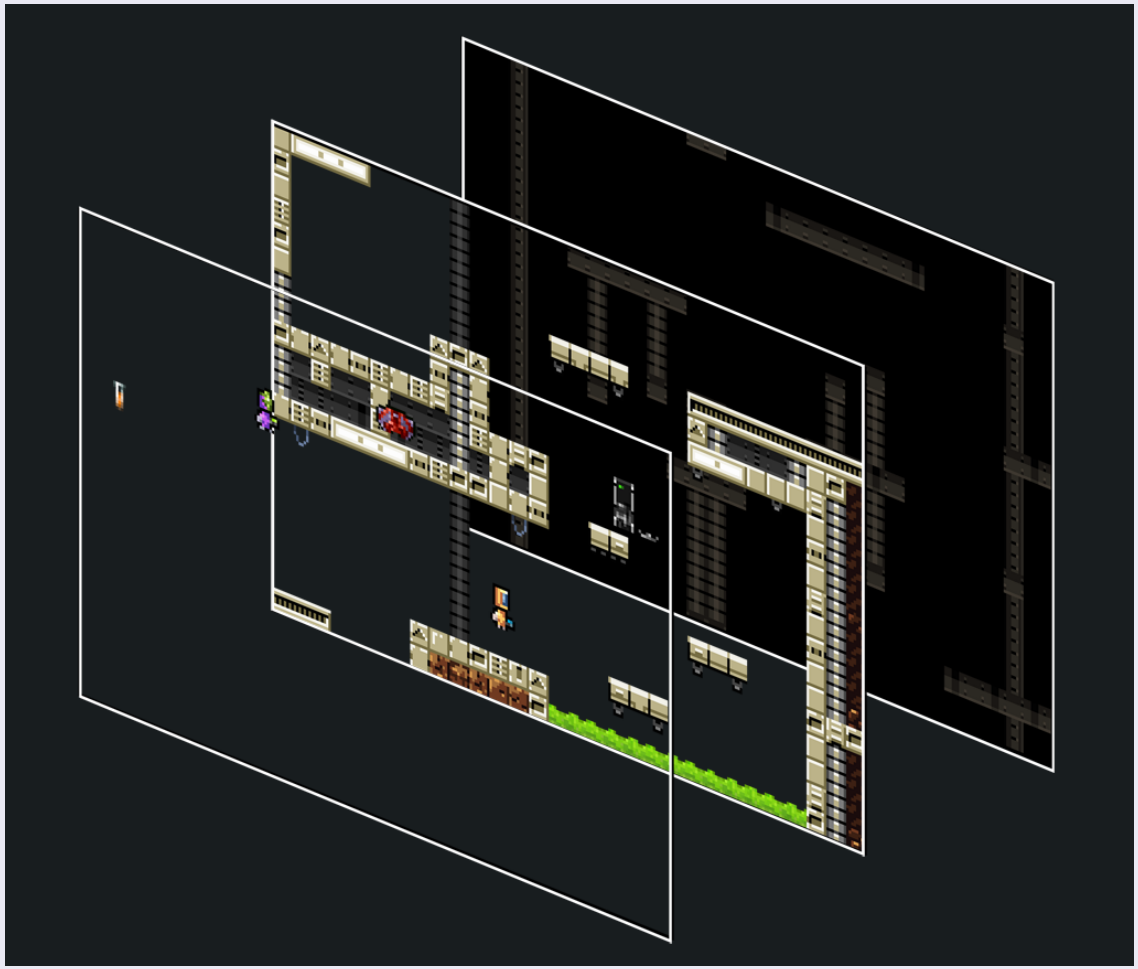
Příklad spritů pro postavičku...



Průhlednost (transparency)

- RGBA obrázky/pixely
- Kreslení bitmap/spritů „přes sebe“

Blending grafiky ve vrstvách



Hra „Pong“ (7)

Obrázek na pozadí, míček a pálka

```
PImage court, ball, paddle;

void setup() {
  ...
  court = loadImage("court.jpg");
  ball = loadImage("ball.png");
  paddle = loadImage("paddle.png");
}
void draw() {
//  background(255, 255, 255);
  background(court);
  ...
}
void ball() {
  image(ball, -ballR, -ballR);
}
void paddle() {
  image(paddle, 0, 0);
}
```