Computação Gráfica e Interfaces

2017-2018

Fernando Birra

Input e Interação

2017-2018

Fernando Birra

Objetivos

- Dispositivos de input
  - dispositivos lógicos
  - dispositivos físicos
- Modos de operação
- Input guiado por eventos
- double buffering
- programação com eventos em WebGL

Interação

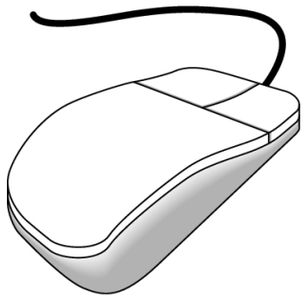
- Em 1963, Ivan Sutherland introduziu o paradigma de interação elementar que caracteriza a computação gráfica interactiva:

  - O utilizador vê um objeto no ecrã
  - O utilizador aponta para (escolhe) o objeto com um dispositivo de input (mouse, tablet, etc)
  - O objeto é manipulado (roda, muda de posição, muda de forma)
  - o processo é repetido

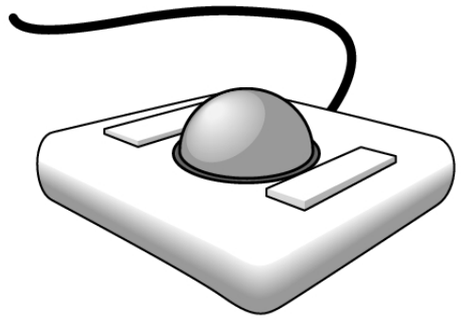
Dispositivos de Input

- Os dispositivos podem ser descritos:
  - pelas suas propriedades físicas
    - rato; teclado; trackball
  - pelo que podem fornecer à aplicação através da API
    - Uma posição
    - um identificador dum objeto
    - um valor numérico

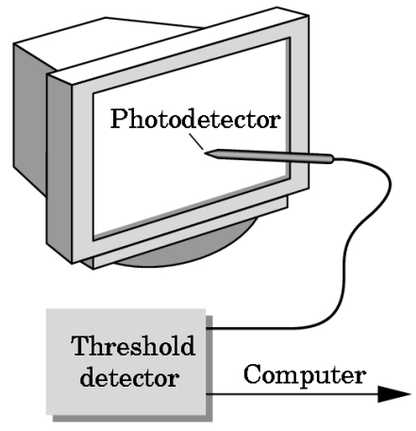
Dispositivos Físicos



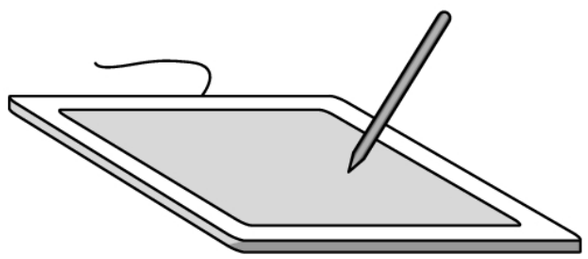
mouse



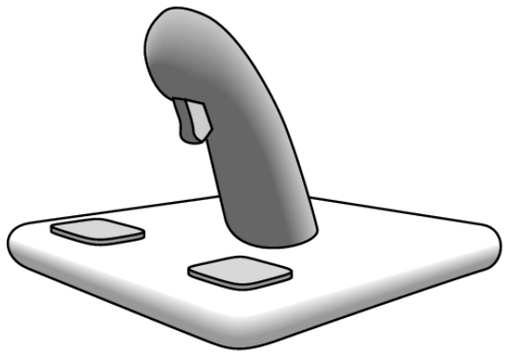
trackball



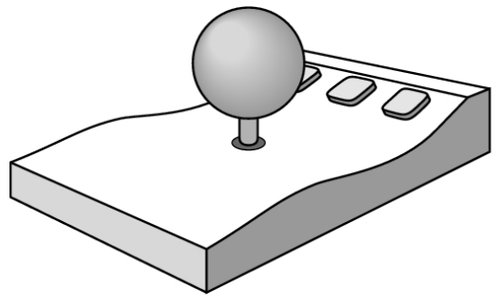
light pen



tablet



joystick



space ball

Dispositivos Absolutos vs Relativos

- Alguns dispositivos são capazes de fornecer uma posição diretamente ao sistema operativo
  - light pen, tablet
- Outros, apenas conseguem fornecer alterações (ou velocidades), cabendo ao sistema operativo a tarefa de integrar essa informação:
  - rotações dos cilindros dum rato mecânico
  - Rotação duma trackball

Dispositivos lógicos

- O input duma aplicação gráfica é de natureza mais variada que o duma aplicação de consola, o qual se resume a números ou caracteres

- Os sistemas PHIGS e GKS definiram 6 tipos de dispositivos lógicos, consoante o tipo de dados que eram capazes de fornecer à aplicação:
  - **Locator**: capaz de fornecer uma posição
  - **Pick**: capaz de retornar o identificador dum objeto
  - **Keyboard**: capaz de retornar uma cadeia de caracteres
  - **Stroke**: capaz de produzir uma sequência de posições
  - **Valuator**: retorna um número real
  - **Choice**: retorna uma opção de entre um leque

Modos

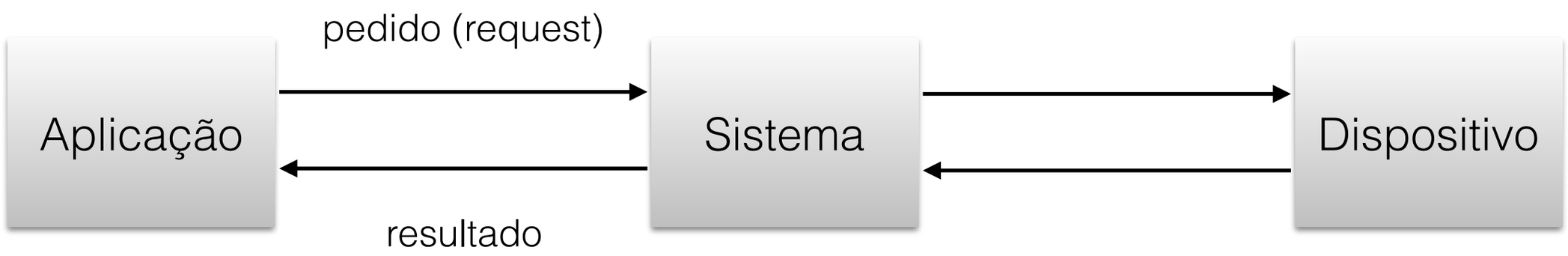
- Os dispositivos de entrada têm a capacidade de desencadear o envio de dados **para o sistema operativo** quando algo acontece:
  - um botão do rato
  - o deslocamento do rato
  - uma tecla que muda de estado no teclado, …
- Quando ativados, os dispositivos de entrada fornecem informação **ao sistema**:
  - o rato reporta uma posição
  - o teclado envia um código da tecla

Modos

- O envio de informação dum dispositivo **para a aplicação** pode ser feito de dois modos distintos:
  - síncrono ou a pedido da aplicação (request mode): a aplicação lê explicitamente o estado do dispositivo (o seu valor)
  - assíncrono, ou em reação a um evento (event mode): a aplicação é avisada de que novos dados estão disponíveis

Modo síncrono (request mode)

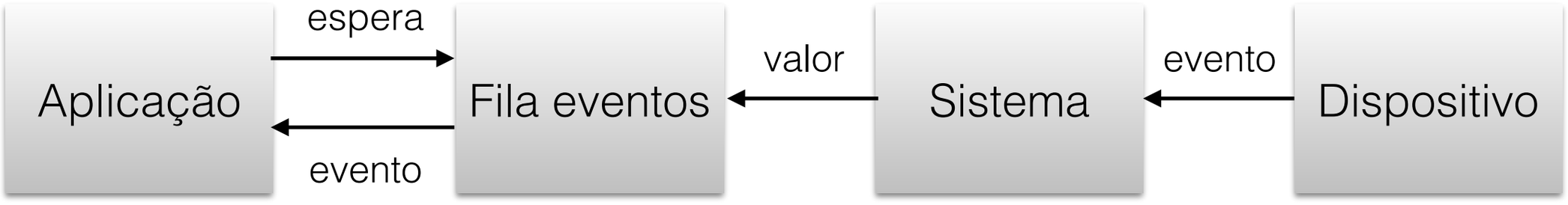
- O input é fornecido a pedido da aplicação
- Um caso típico é o do uso do teclado para entrada de texto
- O resultado só vem porque houve um pedido da aplicação



Modo assíncrono (event mode)

- Múltiplos dispositivos em simultâneo, podendo qualquer um ser acionado pelo utilizador

- Cada ação pode gerar um evento, cujo resultado é colocado numa fila de eventos para serem analisados pela aplicação



Tipos de eventos

- **Janela**: redimensionamento, exposição, minimização, restauro
- **Rato**: click num botão, deslocamento do rato
- **Teclado**: premir ou libertar uma tecla
- **Sistema**: timer, preparar para shutdown ou standby

Callbacks

- A forma de lidar com eventos passa (normalmente) por um mecanismo de callbacks ou event listeners

- Uma callback é um pedaço de código da aplicação registado para o tratamento dum determinado evento em concreto

- Quando o evento acontece, a callback é invocada e a informação do evento é passada a essa função

- A alternativa ao uso de callbacks é a existência dum ciclo (bloqueante ou não) de leitura de eventos da fila

Execução num Browser

1. O browser carrega o HTML

   - descreve a página
   - pode conter shaders
   - carrega os scripts

2. Os scripts são carregados e o seu código executado

3. O browser entra num ciclo à espera que algum evento surja

Execução num Browser

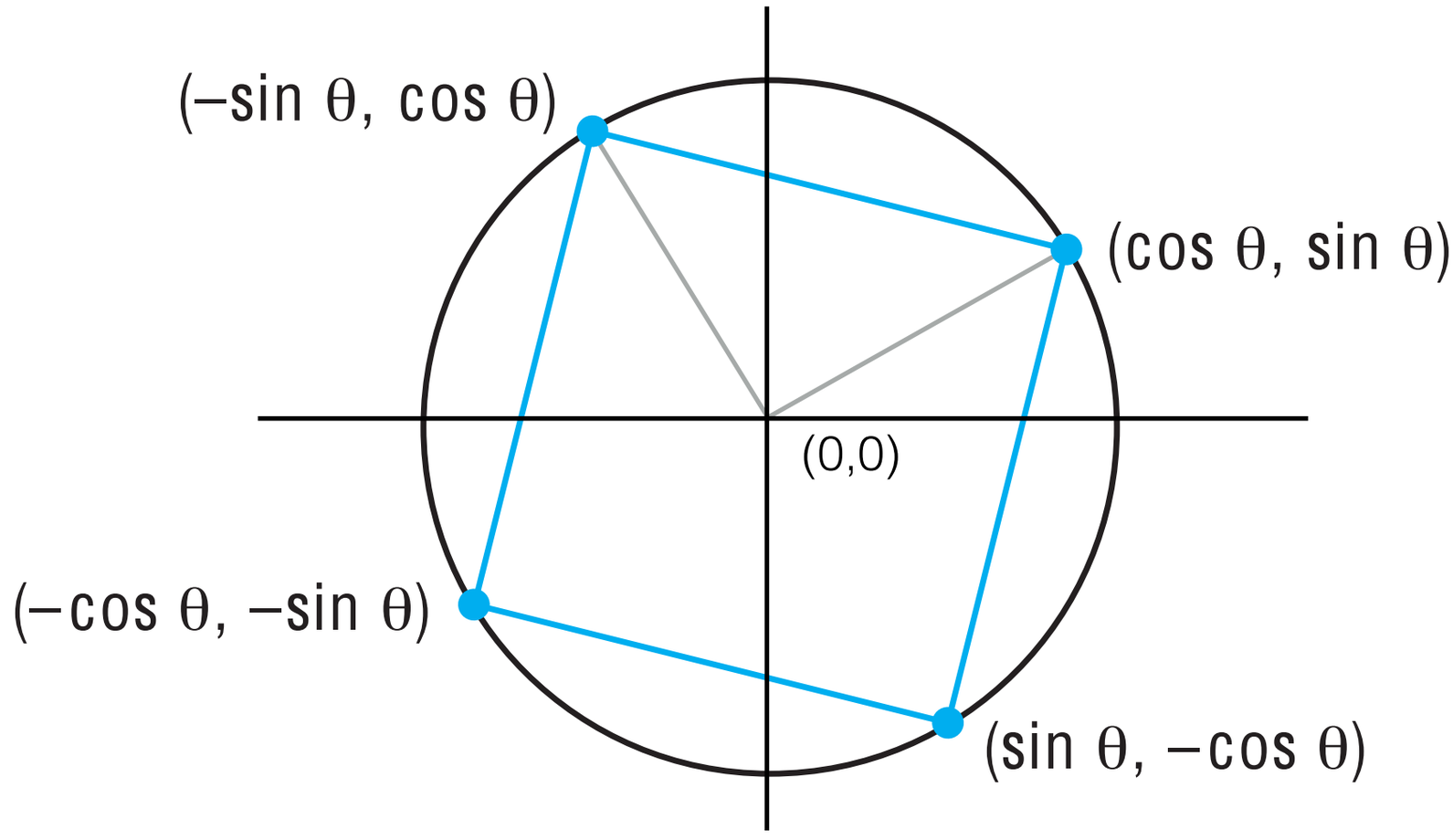
- O código da nossa aplicação é carregado mas nada acontece

- A forma de desencadear a execução da nossa aplicação é associando uma função (init) ao evento **`window.onload`**

- O evento **`window.onload`** é desencadeado automaticamente assim que o browser acaba de carregar toda a informação relativa à página (scripts incluídos)

Exemplo

- Considerando os 4 pontos:



- Animar o quadrado fazendo variar θ

Uma solução (lenta…)

```
var vertices = [
  vec2(0, 1),
  vec2(-1, 0),
  vec2(1, 0),
  vec2(0, -1)
];
```

```
for(var theta=0.0; theta < thetaMax; theta += theta) {
  vertices[0] = vec2(Math.sin(theta), Math.cos(theta));
  vertices[1] = vec2(Math.sin(theta), -Math.cos(theta));
  vertices[2] = vec2(-Math.sin(theta), Math.cos(theta));
  vertices[3] = vec2(-Math.sin(theta), -Math.cos(theta));

  gl.bufferSubData(. . .

  render();
}
```

todas as operações são efetuadas no CPU, sequencialmente, obrigando ao envio dos dados a cada *frame*

Uma alternativa melhor

- Enviar os vértices iniciais para o vertex shader
- Enviar θ como uma variável *uniform*
- Calcular os vértices efetivos no vertex shader
- Desenhar e repetir o processo fazendo variar θ

Função render

```
var thetaLoc = gl.getUniformLocation(program, “theta”);


function render()
{
    gl.clear(gl.COLOR_BUFFER_BIT);

    theta += 0.1;

    gl.uniform1f(thetaLoc, theta);
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

    render();
}
```

Vertex shader

```
attribute vec4 vPosition;
uniform float theta;

void main()
{
    float s = sin( theta );
    float c = cos( theta );

    gl_Position.x = -s * vPosition.y + c * vPosition.x;
    gl_Position.y =  s * vPosition.x + c * vPosition.y;
    gl_Position.z = 0.0;
    gl_Position.w = 1.0;
}
```

Double Buffering

- Enquanto o programa desenha o quadrado, fá-lo sempre num buffer que não está a ser mostrado

- O frame buffer é desdobrado em dois buffers (o front buffer - visível - e o back buffer - escondido)

- O browser usa *double buffering*

  - Está sempre a mostrar o conteúdo do *front buffer*

  - As atualizações são sempre feitas no *back buffer*

  - No final troca os papéis dos 2 buffers (swap)

- Impede a visualização parcial dum quadro (*frame)*

Interval Timer

- Permite executar uma função depois de decorrido um determinado tempo (em milisegundos)
  - com a consequência de gerar uma troca de buffers

```
function render(){
   …
   // render code
   …
}
setInterval(render, interval);
```

- Um intervalo de 0 provocará eventos (e trocas de buffer) tão rapidamente quando possível

requestAnimationFrame

```
function render()
{
   gl.clear(gl.COLOR_BUFFER_BIT);

   theta += 0.1;

   gl.uniform1f(thetaLoc, theta);
   gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

   requestAnimationFrame(render);
}
```

A cadência da troca dos buffers é determinada pelo browser (cerca de 60Hz)

Nota: *requestAnimFrame* (definida em webgl-utils.js) chama a função *requestAnimationF*rame se o browser suportar, caso contrário simula o seu funcionamento.

Com controlo do intervalo

```
function render()
{
   setTimeout(function() {

      requestAnimFrame(render);
      gl.clear(gl.COLOR_BUFFER_BIT);

      theta += 0.1;

      gl.uniform1f(thetaLoc, theta);
      gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

   }, interval);
}
```

A cadência da troca dos buffers é determinada por *interval*, desde que não ultrapasse os 60 fps.