

Input e Interação

Objetivos

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

Interação

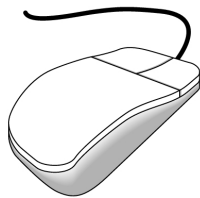
- Em 1963, Ivan Sutherland introduziu o paradigma de interação elementar que caracteriza a computação gráfica interactiva:
 - O utilizador vê um objeto no ecrã
 - O utilizador aponta para (escolhe) o objeto com um dispositivo de input (mouse, tablet, etc)
 - O objeto é manipulado (roda, muda de posição, muda de forma)
 - o processo é repetido



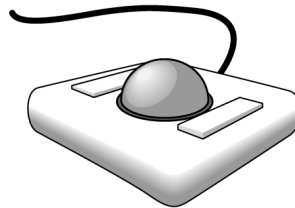
Dispositivos de Input

- 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

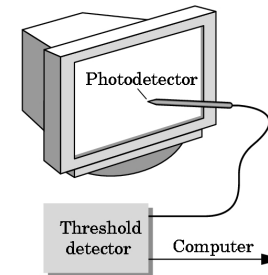
Dispositivos Físicos



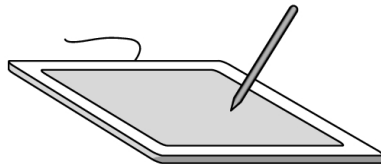
mouse



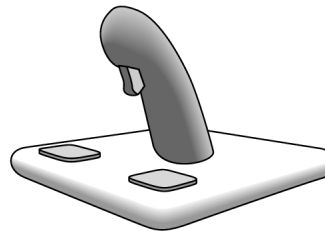
trackball



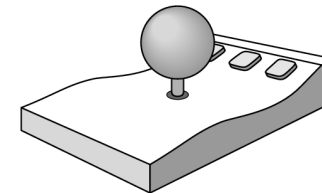
light pen



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 dum trackball

Dispositivos lógicos

- O input numa aplicação gráfica é de natureza mais variada que o numa 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

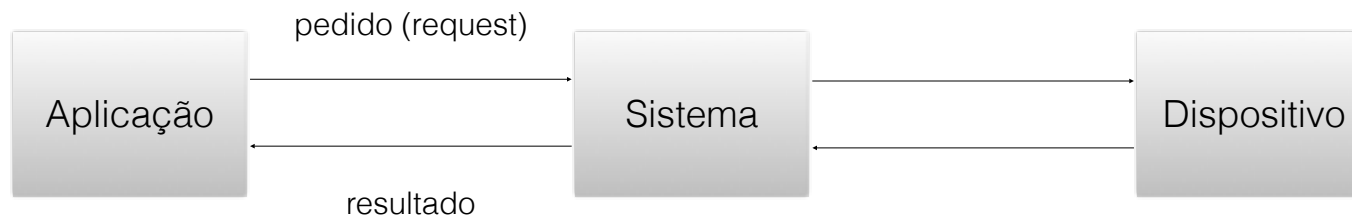
- Os dispositivos de entrada têm a capacidade de desencadear o envio de dados **para o sistema operativo** quando algo acontece:
 - um botão do rato
 - o deslocamento do rato
 - uma tecla que muda de estado no teclado, ...
- Quando ativados, os dispositivos de entrada fornecem informação **ao sistema**:
 - 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 algum evento ocorreu e de que novos dados estão disponíveis

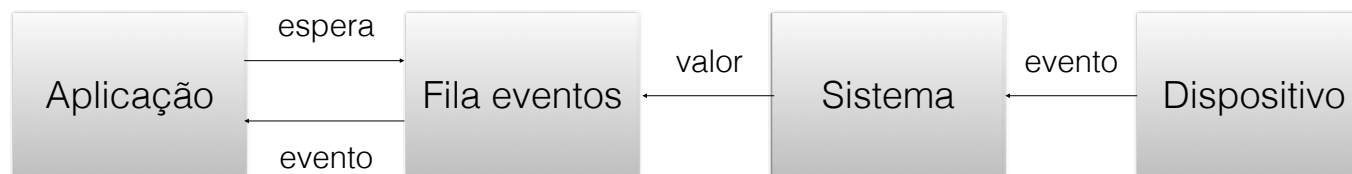
Modo síncrono (request mode)

- O input é fornecido a pedido da aplicação
- Um caso típico é o do uso do teclado para entrada de texto (em aplicações não gráficas)
- O resultado só vem porque houve um pedido da aplicação
- Apenas se permite a utilização dum dispositivo de cada vez



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, botão para baixo, botão para cima
- **Teclado:** premir ou libertar uma tecla, introdução dum carácter
- **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 acerca do evento é passada a essa função (posição do cursor, modificadores do teclado, botão premido, etc.)
- A alternativa ao uso de callbacks é a existência dum ciclo (bloqueante ou não) de leitura de eventos da fila

Execução num Browser

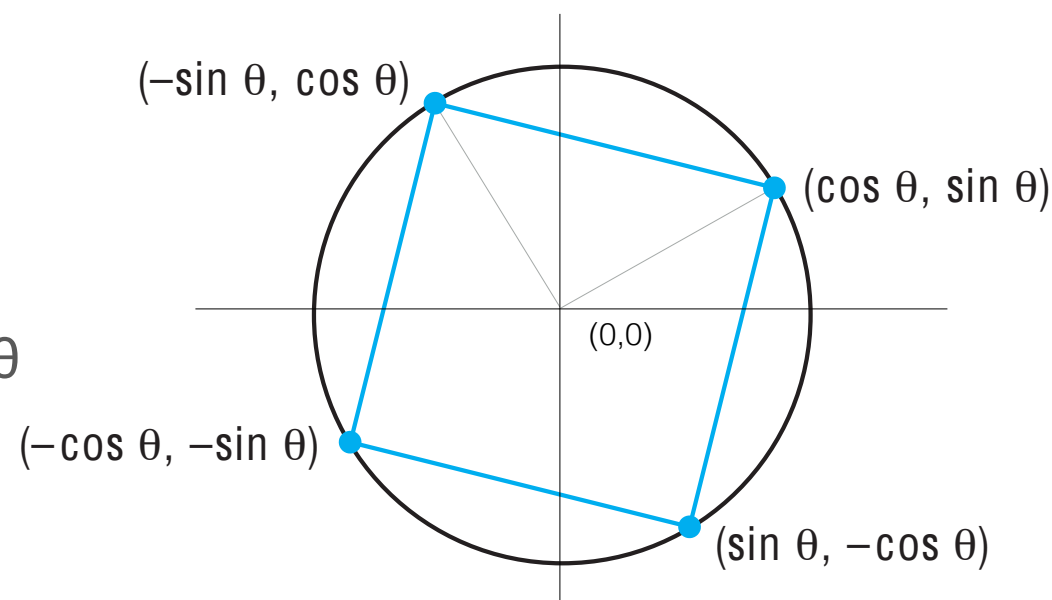
1. O browser carrega o HTML
 - descreve a página
 - pode conter shaders
 - carrega os scripts
2. Os scripts são carregados e o seu código executado
3. O browser entra num ciclo à espera que algum evento surja

Execução num Browser

- O código da nossa aplicação é carregado mas nada acontece
- Uma forma de desencadear a execução da nossa aplicação é colocando uma chamada no script para a função principal ou associando uma função principal 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 += dtheta) {  
    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(. . . . .  
  
    animate();  
}
```

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 animate

```
var u_theta = gl.getUniformLocation(program, "u_theta");
```

```
function animate()  
{  
    gl.clear(gl.COLOR_BUFFER_BIT);  
  
    theta += 0.1;  
  
    gl.uniform1f(u_theta, theta);  
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);  
  
    animate();  
}
```

Vertex shader

```
attribute vec4 a_position;  
uniform float u_theta;  
  
void main()  
{  
    float s = sin( u_theta );  
    float c = cos( u_theta );  
  
    gl_Position.x = -s * a_position.y + c * a_position.x;  
    gl_Position.y =  s * a_position.x + c * a_position.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*)

Função animate

```
var u_theta = gl.getUniformLocation(program, "u_theta");
```

```
function animate()  
{  
    gl.clear(gl.COLOR_BUFFER_BIT);  
  
    theta += 0.1;  
  
    gl.uniform1f(u_theta, theta);  
    gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);  
  
    animate();  
}
```

Para além de ser recursivo (e esgotar o stack), nada nos garantiria que veríamos todos os quadros (frames) gerados! Na realidade não veríamos rigorosamente nada!

Buffer Swap

- Os browsers atualizam o ecrã a uma taxa de cerca de 60Hz
 - redesenham o front buffer
 - não efetuam a troca de buffers
- A troca dos buffers é efetuada em resposta a um evento
- Há duas opções:
 - interval timer
 - requestAnimationFrame

Interval Timer

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

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

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

requestAnimationFrame

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

    theta += 0.1;

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

    window.requestAnimationFrame(animate);
}
```

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

Com controlo do intervalo

```
function animate()
{
    setTimeout(function() {

        requestAnimationFrame(animate);
        gl.clear(gl.COLOR_BUFFER_BIT);

        theta += 0.1;

        gl.uniform1f(u_theta, 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.

Event Listeners

Objetivos

- Aprender a criar programas interativos usando *event listeners (callbacks)*
 - Botões
 - Menus
 - Mouse
 - Teclado
 - Reshape (janela)

Exemplo: Botão para controlar direção de rotação

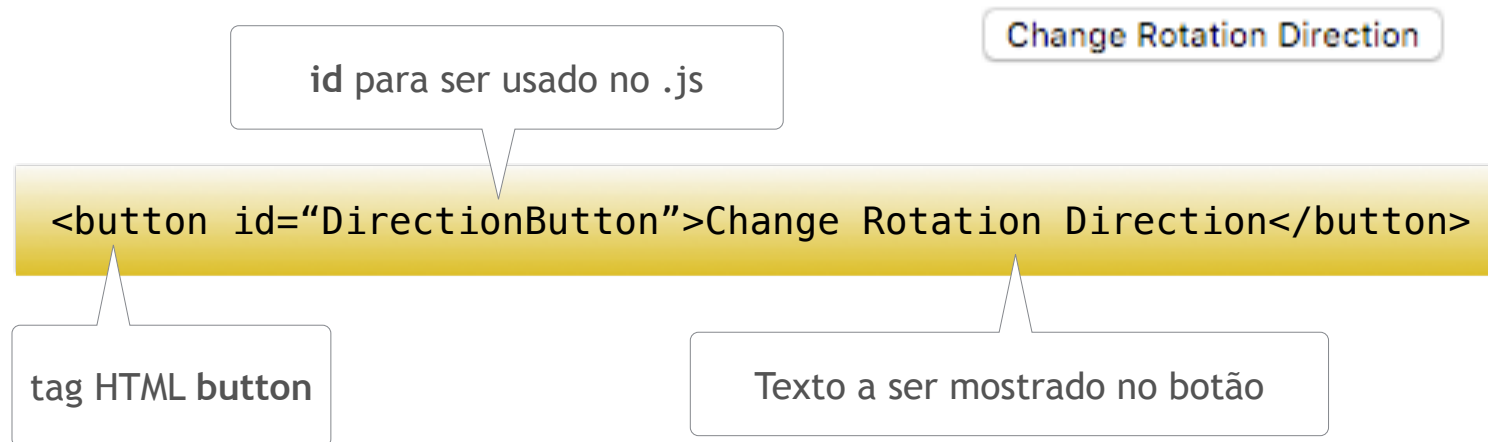
- Começamos por adicionar um botão ao exemplo do quadrado, para controlar a direção de rotação do mesmo
- Na função `render` podemos usar uma variável booleana *direction* que controlará o incremento ou decremento do ângulo

```
let direction = true; // global initialisation

animate() {
  ...
  if(direction) theta += 0.1;
  else theta -= 0.1;
  ...
}
```

○ botão

- No ficheiro HTML:



Fazendo um click no botão gera um evento `click`

Event Listener do botão

- Sem nenhum *event listener* definido, o evento ocorre mas é ignorado...
- Distintas formas para adicionar o tratamento do evento:

```
document.getElementById("DirectionButton").onclick =  
function() {  
    direction = !direction;  
};
```

```
document.getElementById("DirectionButton").addEventListener("click"  
, function() {  
    direction = !direction;  
});
```

A evitar! Código no HTML!

```
<button ... click="direction=!direction">...</button>
```

Controlo da velocidade



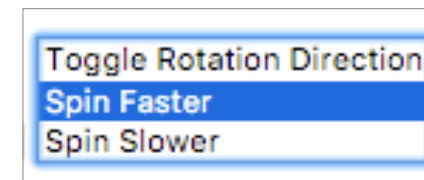
```
let delay = 100;

function animate()
{
    setTimeout(function() {
        requestAnimationFrame(animate);
        gl.clear(gl.COLOR_BUFFER_BIT);
        theta += (direction ? 0.1 : -0.1);
        gl.uniform1f(u_theta, theta);
        gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
    }, delay);
}
```


Menu para controlo da velocidade

- O elemento HTML `select` oferece um leque de escolhas (menu)
- Cada entrada no menu é um elemento `option` com um inteiro `value` retornado no evento click

```
<select id="mymenu" size="3">
  <option value="0">Toggle Rotation Direction</option>
  <option value="1">Spin Faster</option>
  <option value="2">Spin Slower</option>
</select>
```



Event Listener do menu

```
var m = document.getElementById("mymenu");  
m.addEventListener("click", function() {  
    switch (m.selectedIndex) {  
        case 0:  
            direction = !direction;  
            break;  
        case 1:  
            delay /= 2.0;  
            break;  
        case 2:  
            delay *= 2.0;  
            break;  
    }  
});
```

Controlo através do teclado

objeto
window
representa o
conteúdo
todo do
browser

```
window.addEventListener("keydown", function(event) {  
    switch (event.keyCode) {  
        case 49: // '1' key  
            direction = !direction;  
            break;  
        case 50: // '2' key  
            delay /= 2.0;  
            break;  
        case 51: // '3' key  
            delay *= 2.0;  
            break;  
    }  
});
```

Variante (sem keycodes)

```
window.onkeydown = function(event) {  
    const key = String.fromCharCode(event.keyCode);  
    switch (key) {  
        case '1':  
            direction = !direction;  
            break;  
        case '2':  
            delay /= 2.0;  
            break;  
        case '3':  
            delay *= 2.0;  
            break;  
    }  
};
```

Elemento Slider

- Elemento input de tipo range permite fornecer à aplicação um valor num intervalo


```

<div>
  speed 0% <input id="slide" type="range" min="0" max="100"
  step="10" value = "50"/> 100%
</div>

```

deslocamento
necessário para
gerar evento

valor inicial



Event Listener do slider

id do slider

```
document.getElementById("slide").onchange =  
    function(event) {  
        delay = event.srcElement.value;  
    };
```

Também se poderia ter usado **oninput** e, nesse caso, os eventos são gerados continuamente.

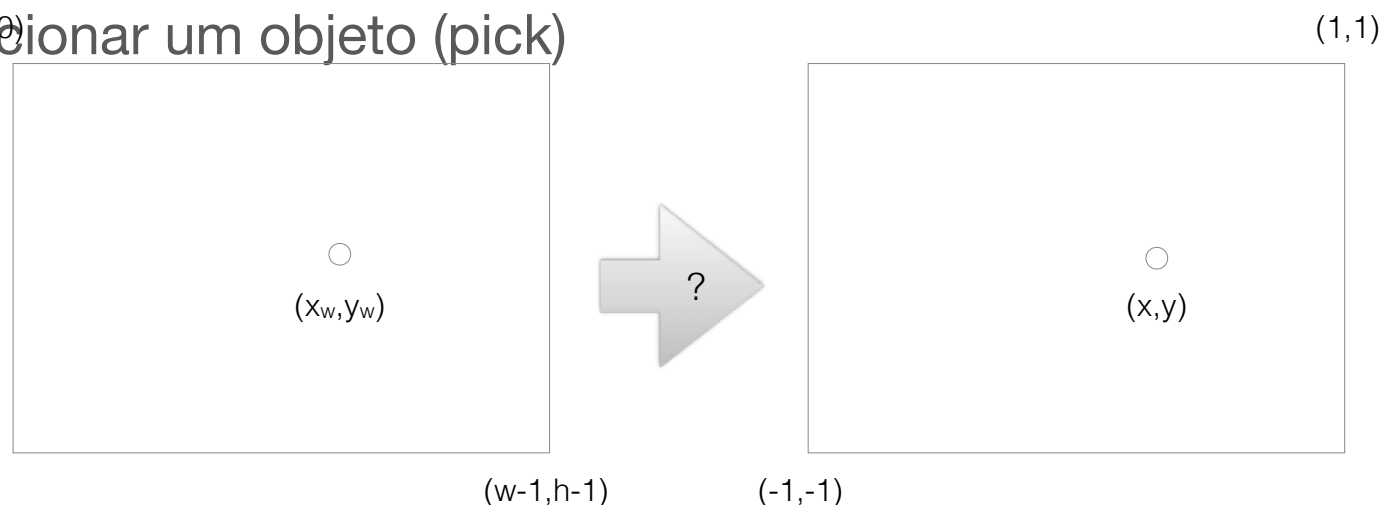
Aquisição de posições

- É necessário converter das coordenadas do ecrã, fornecidas num evento do rato, por exemplo, para coordenadas do modelo!



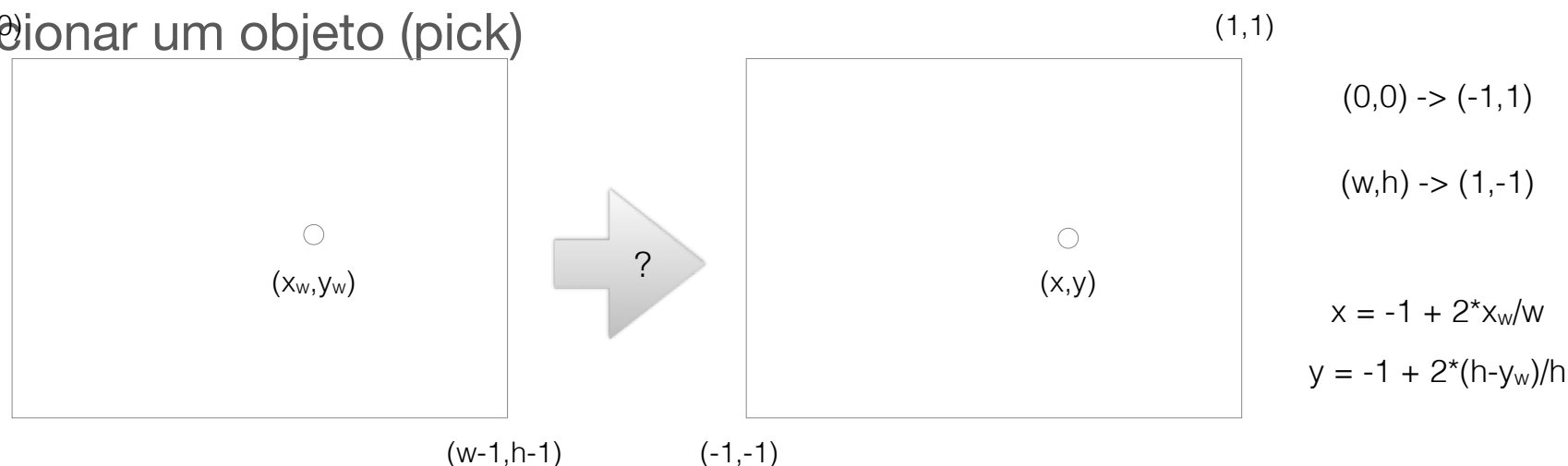
Aquisição de posições

- Esta transformação pode ser importante para:
 - reponder a eventos de mudança das dimensões da janela
 - desenhar novos elementos no ecrã, na posição fornecida pelo utilizador
 - selecionar um objeto (pick)



Aquisição de posições

- Esta transformação pode ser importante para:
 - reponder a eventos de mudança das dimensões da janela
 - desenhar novos elementos no ecrã, na posição fornecida pelo utilizador
 - selecionar um objeto (pick)



Conversão de coordenadas

- O canvas, especificado no HTML, tem dimensões `canvas.width` x `canvas.height`
- As coordenadas retornadas referentes ao campo superior esquerdo do canvas são: `event.offsetX` e `event.offsetY`

```
// add a vertex to GPU for each click
```

```
canvas.addEventListener("click", function() {  
    gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);  
    const t = vec2(-1 + 2*event.offsetX/canvas.width,  
                  -1 + 2*(canvas.height-event.offsetY)/canvas.height);  
    gl.bufferSubData(gl.ARRAY_BUFFER, sizeof['vec2']*index, flatten(t));  
    index++;  
});
```

Eventos de janelas

- Os eventos podem ser gerados por ações que afetam a janela do canvas
 - mover ou expor uma janela
 - redimensionar uma janela
 - abrir uma janela
 - minimizar/restaurar uma janela
- Há callbacks por omissão para cada um destes eventos

Evento *onresize*

- Um evento *onresize* ocorre quando se dá a alteração das dimensões duma janela
- Em resposta é necessário:
 - atualizar o conteúdo da mesma. Opções:
 - Mostrar os mesmos objetos mas com diferente tamanho
 - Mostrar mais ou menos objetos mantendo o tamanho
 - Na maior parte das vezes queremos manter as proporções

Evento *onresize*

- As dimensões da nova janela podem ser acedidas através de `window.innerWidth` e `window.innerHeight`
- Usa-se `window.innerWidth` e `window.innerHeight` para alterar `canvas.width` e `canvas.height`
- Exemplo: Manter um canvas quadrado, maximizando a sua área

Exemplo: Mantendo o canvas quadrado

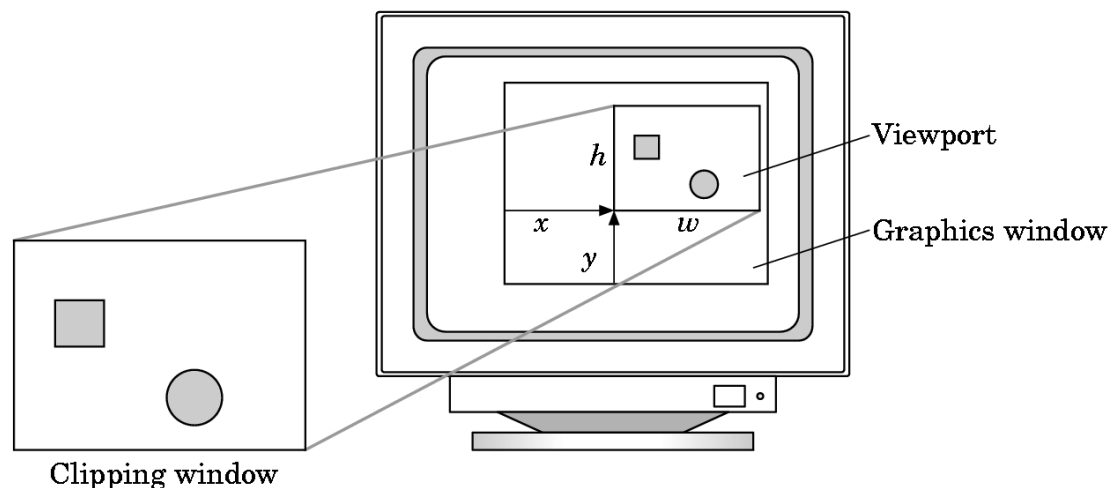
```
window.onresize = function() {  
    const height = window.innerHeight;  
    const width = window.innerWidth;  
    const s = Math.min(width, height);  
    canvas.width = s;  
    canvas.height = s;  
    gl.viewport(0,0,s,s);  
};
```

Assume-se neste exemplo que o canvas é inicialmente quadrado e que o objetivo é maximizar a sua área, mantendo o seu formato.

a função `viewport` permite definir a origem e a dimensão da área do canvas que é efetivamente usada para visualização dos gráficos (visor)

Viewport

- Uma aplicação não é obrigada a usar toda a área do canvas para a imagem:
`gl.viewport(x,y,w,h)`
- Os valores são dados em pixels (coordenadas da janela)



(x,y) são as coordenadas do canto inferior esquerdo do visor (*Viewport*)

w e h são, respetivamente, a largura e a altura do visor.