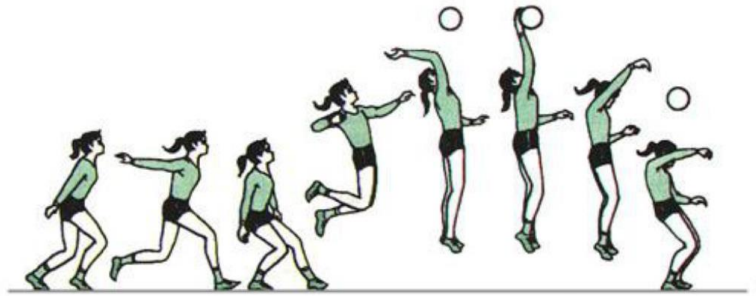
Objectives

- **Animation**
  - **Double Buffering**
  - **Time Delay**
- Event Input Programming
  - the Basic Input Devices
  - Input Modes
  - Programming event input with WebGL
- Example： Rotate Square
  - Simple but Slow Method~JS Programming
  - Better way ~Vertext Shader Programming
- More input techniques
  - Use the mouse to give locations
  - CAD-like Examples

动画和交互

- 前面的示例程序只能静态地显示对象。
  - 首先，用户通过应用程序输入描述场景对象，
  - 然后，应用程序向GPU发送顶点数据，
  - 最后，GPU绘制并显示一帧画面。
- 大多数应用中需要动态地显示对象（对象改变位置颜色形状等）
  - 没有用户输入，画面会随时间变化---动画
  - 由用户输入，画面根据输入作相应地变化---交互
  - 涉及多帧画面的绘制

动画



- 动画需要刷新屏幕绘制动态变化的画面。
- 帧频：每秒刷新多少关键帧（fps: **frames per second**）
- 帧频是每秒24帧以上的运动序列，才能达到画面连贯的动画

  （动画帧频一般是每秒24~150帧（赫兹））
  - 如果帧频<24，则可能有帧断裂或帧飘移
  - 如果帧频>150，则可能有帧拖延效果（晕炫视觉，拖影）

注：动画帧频与显示器刷新频率不同（每秒60赫兹以上，屏幕不闪烁）

动画类型

- 对每帧，重新绘制整个画面 比 判断哪些部分需要重新绘制 更容易!

- 动画有两种:

1）逐帧动画:

- 离线生成，实时回放。
- 如：电影，电视，动漫。

2）实时动画:

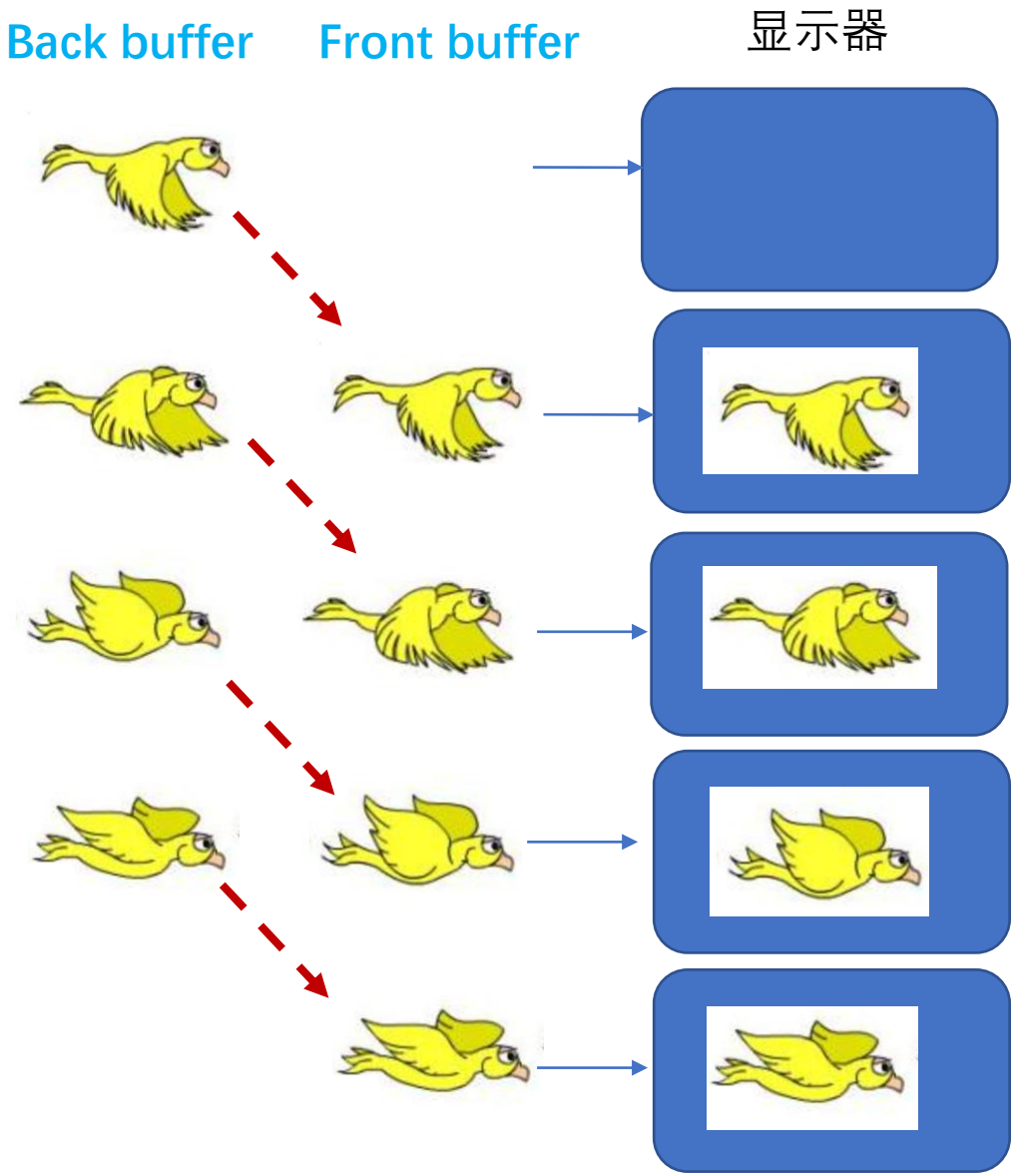
- 在线生成，立即播放。
- 如：游戏，交互应用。

Double Buffering（双缓存机制）

- 单帧缓存：
  - 一个颜色帧缓存，可读可写。
  - 若创建一帧时间>屏幕一个刷新周期，则导致帧飘移或帧破裂。
- 双帧缓存：
  - 两个颜色帧缓存，一个用于写新帧，另一个用于显示。在两帧间隙时间完成缓存的切换。
  - 双缓存技术防止了画面的部分显示（即帧频过低产生的“帧破裂”问题）

双帧绘制机制

- 显示器读取前帧缓存Front buffer，绘制屏幕。
- GPU绘制画面到后帧缓存Back buffer。
- 后帧绘制好后，需要进行帧交换：将back buffer 内容复制到front buffer中进行显示
  - *front buffer：Always display*
  - *back buffer ：Rendering into*



Triggering a Buffer Swap（帧交换）

- 帧交换：
  动画需要不断进行帧交换，将更新后的后帧内容切换为前帧去显示。

- Need a buffer swap需要一个事件来触发帧交换。两种方式
  - **Interval timer 设置时延**
    如果创建一帧的时间非常接近刷新周期的整数倍，则容易出现不规则动画帧率的问题，如有的帧创建时间短一点，有些长一点，导致动画帧率不稳定，弥补方法是在程序绘制代码中加入少许时延。

  - **requestAnimFrame 请求动画帧**
    完成后帧的绘制后，触发/请求帧交换，后帧变成前帧进行显示

帧交换方式1：Interval Timer

- **WebGL:** Javascript function, independent to browser
  每个浏览器中可能有区别，难以得到平滑动画显示（一般不直接用）

```
setInterval(render, interval);
```

Executes a function（render） after a specified number of milliseconds毫秒(interval)

- Also generates a buffer swap
- Note an interval of 0：generates buffer swaps as fast as possible

帧交换方式2：requestAnimFrame

- **WebGL:** More browser support

```
requestAnimFrame(render);
```

//请求浏览器在下一次屏幕刷新时，显示render函数中绘制的内容。

帧交换3：Combination method

- 前面两种的结合方式：考虑延迟和请求

```
//后帧绘制好，再隔100毫秒，后帧变前帧后进行显示。

delay=100;  //100毫秒= 0.1秒

setTimeout( function ()  { requestAnimFrame( render );},  delay );
```

动画实现~WEBGL示例（演示程序）

//请求帧方式 : RotateSquare_RequestAnimFrame.js:

window.requestAnimFrame(render);

~帧画面绘制简单，使动画帧交换频繁(帧频高),有帧拖延（晕炫视觉/拖影）

//组合方式（延时+请求）:RotateSquare_setTimeOut.js:

setTimeout(function () {requestAnimFrame(render);}, 100);

~前帧绘制后等待100毫秒才绘制后帧，使帧频降低，减弱或避免了帧拖延

Objectives

- Animation
  - Double Buffering
  - Time Delay
- Event Input Programming
  - the Basic Input Devices
  - Input Modes
  - Programming event input with WebGL
- Example：Rotate Square
  - Simple but Slow Method~JS Programming
  - Better way ~Vertext Shader Programming
- More input techniques

the basic interactive paradigm（交互范式）

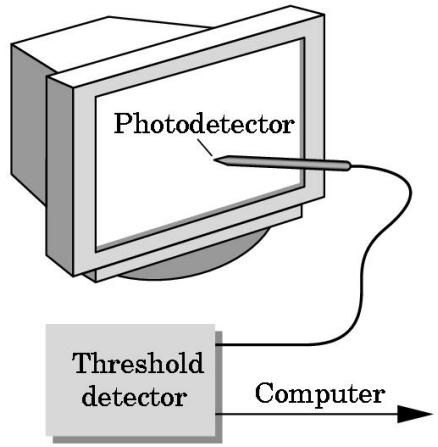
Ivan Sutherland (MIT 1963) established the basic interactive paradigm (Project Sketchpad**)**that characterizes interactive computer graphics:

- User sees an object on the display
- User points to (*picks*) the object with an input device (light pen, mouse, trackball)
- Object changes (moves移动, rotates旋转, morphs变形)
- Repeat

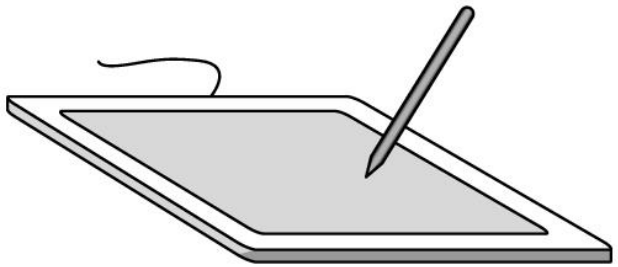
Graphical Physical Devices

mouse

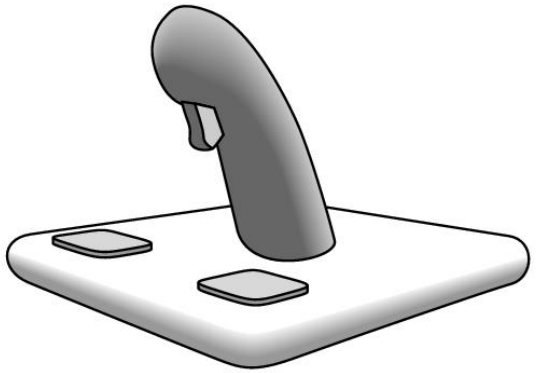
trackball



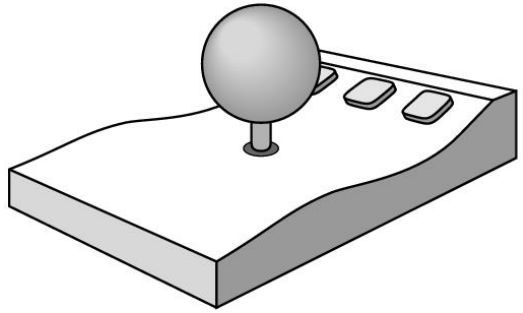
light pen



data tablet



joy stick



space ball

Graphical Logical Devices

- Consider the C and C++ code
  - C++: **`cin >> x;`**
  - C: **`scanf ("%d", &x);`**
- What is the input device?
  - Can't tell from the code, could be keyboard, file, output from another program.
  - The code provides ***logical input***: A number (an **`int`**) is returned to the program regardless of the physical device
- Graphical input is more varied than input to standard programs which is usually numbers, characters, or bits

  （图形输入比标准程序的输入更多样化，通常是数字，字符，或位元）

Graphical Logical Input

lec4-演示和例程\交互演示\逻辑输入设备\封面1.SWF

- Two older APIs (GKS, PHIGS) defined six types of logical input
  - **Locator**: return a position
    - 定位设备：用户自己程序中完成从屏幕坐标到世界坐标的转换
  - **Pick**: return ID of an object
    - 拾取设备：同定位器locator一样的物理设备来实现，但有独立的程序软件接口
  - **Keyboard**: return strings of characters
    - 字符串设备：键盘输入
  - **Stroke**: return array of positions
    - 笔划设备：定位设备的多次使用，返回一个位置数组（开始和结束）
  - **Valuator**: return floating point number
    - 定值设备：构件来完成，如：滑动条，单选按钮
  - **Choice**: return one of n items
    - 选择设备： 菜单、滚动条和图形按钮

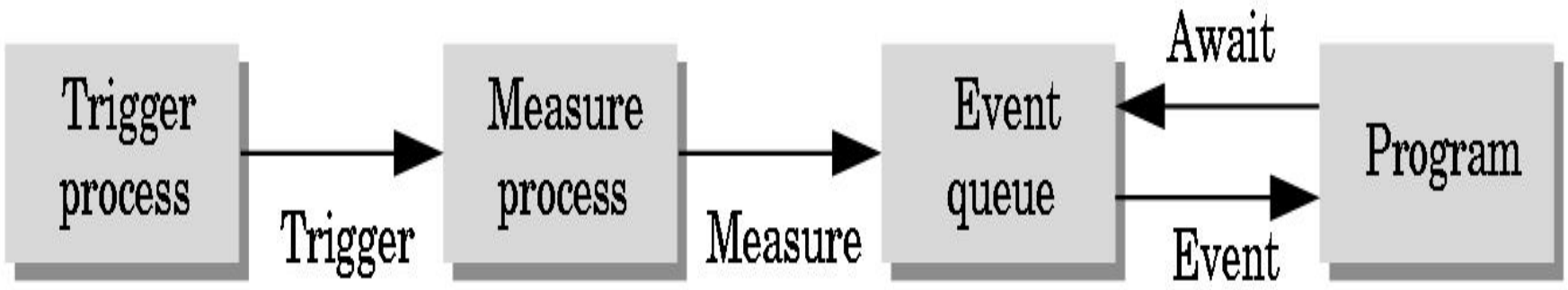
Input Progress输入过程

- Input devices contain a *trigger* 触发器which can be used to send a signal to the operating system
  - Button on mouse
  - Pressing or releasing a key

- When triggered, input devices return *information (their measure 度量值)* to the system
  - Mouse returns position information
  - Keyboard returns ASCII code

Input Modes~ Event Mode

- **Most systems have more than one input device, each of which can be triggered at an arbitrary time by a user**
- Each trigger generates an event whose measure is put in an *event queue* which can be examined by the user program



Event Types

- **Window**: resize, expose, iconify
- **Mouse**: click or release one or more buttons
- **Motion**: move mouse
- **Keyboard**: press or release a key
- **Idle**: nonevent
  - Define what should be done if no other event is in queue

Callbacks function回调函数

- Programming interface编程接口 for event-driven input
  uses ***callback functions（回调函数）*** or ***event listeners（事件监听器）***
  - Define a callback for each event the graphics system recognizes

  对图形系统可识别的每个输入事件都定义了一个回调函数

  - The callback function is executed when the event occurs

  当该事件发生时，回调函数被执行

  - Browsers enters an event loop and responds to those events for which it has callbacks registered

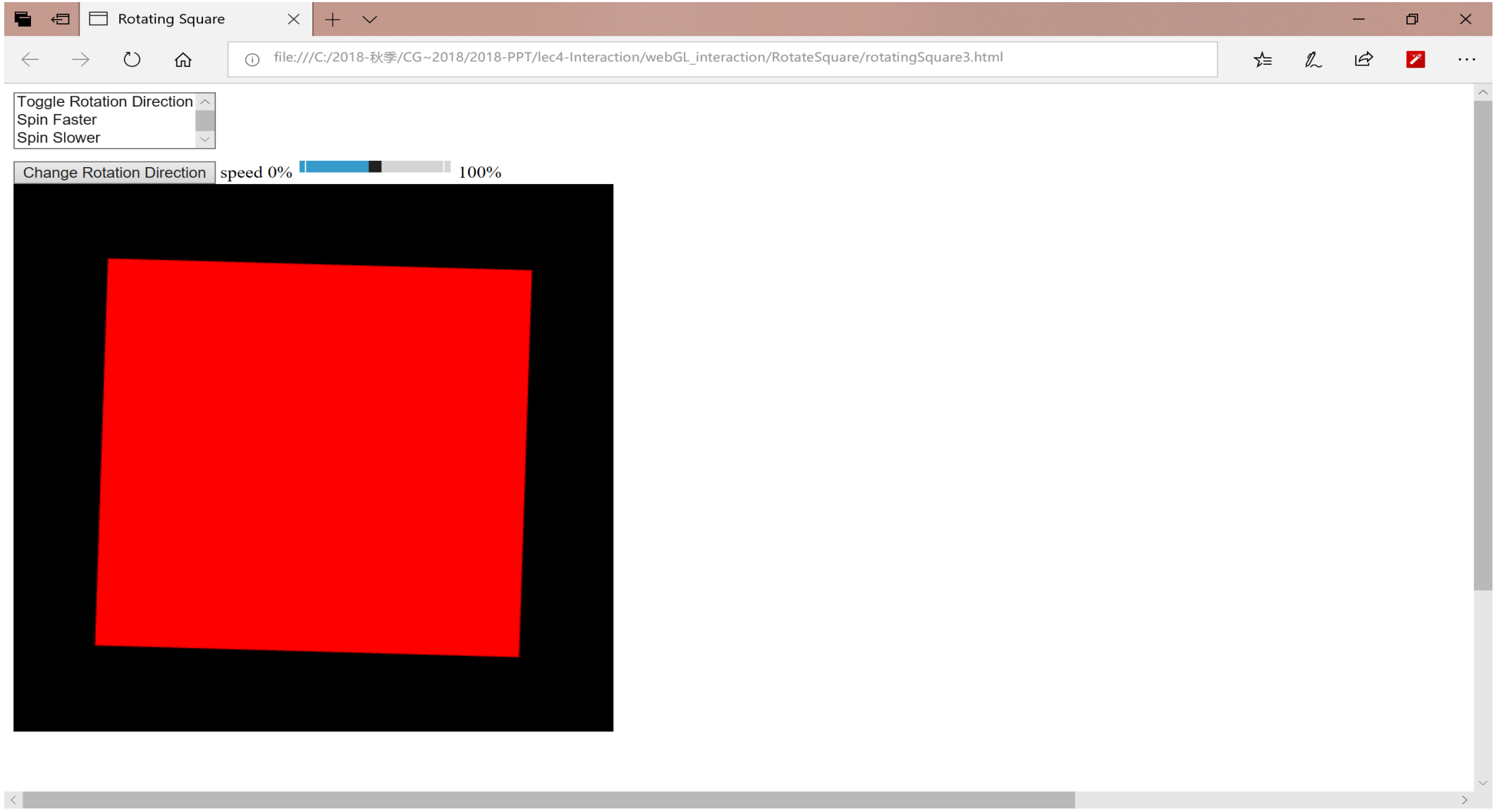
  浏览器进入一个事件循环，会对注册了的回调函数做出响应

WebGL event-driven programming

- Browser is in an event loop and waits for an event
- Target，Event types and their callbacks functions:
  - **Window: load，keydown, Reshape, …**
    例如：window.onload = function init() {……}
  - **Button: click，mousedown**
  - **Menu: click**
  - **Slider: change**
  - **Mouse: mousedown**

Programming event input with WebGL

rotatingSquare



1.Adding a Button


```
<button id="DirectionButton">
        Change Rotation Direction
</button>
```

- In the HTML file
  - Uses HTML button tag
  - id gives an identifier we can use in JS file
  - Text “Change Rotation Direction” displayed in button
- Clicking on button generates a click event
- Note we are using default style and could use CSS or jQuery to get a prettier button

Declare variable “direction”

```
var direction = true; // global initialization
```

- In the render function we can use a var direction which is true or false to add or subtract a constant to the angle

```
// in render()
if(direction)    theta += 0.1; else theta -= 0.1;
```

Button Event Listener

- **We still need to define the listener: no listener and the event occurs but is ignored**
- **Two forms for event listener in JS file, choose one:**

```
var myButton = document.getElementById("DirectionButton");

myButton.addEventListener("click", function() {
    direction = !direction;
});
```

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

onclick Variants（onclick操作的变异）

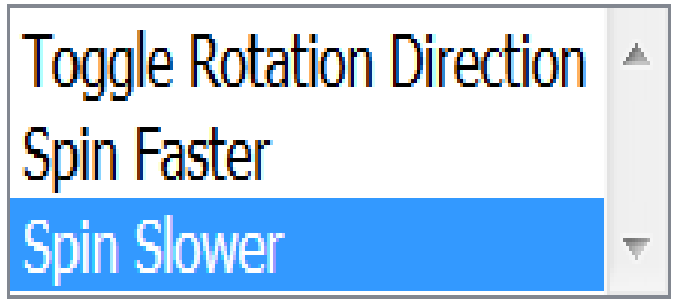
- More onclick **variants**：更多实现，下面列了三种

```
myButton.addEventListener(“click", function(event) {
  if (event.button == 0) { direction = !direction; }
}); //Event.button=0表示是鼠标左键
```

```
myButton.addEventListener("click", function(event) {
  if (event.shiftKey == 0) { direction = !direction; }
});//Enent.shiftkey:表示按下shift键
```

```
<button onclick="direction = !direction"></button>
```

2.Menus




```
<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>
```

- Use the HTML select element
- Each entry in the menu is an option element with an integer value returned by click event

Menu Listener

```
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;
       }
});
```

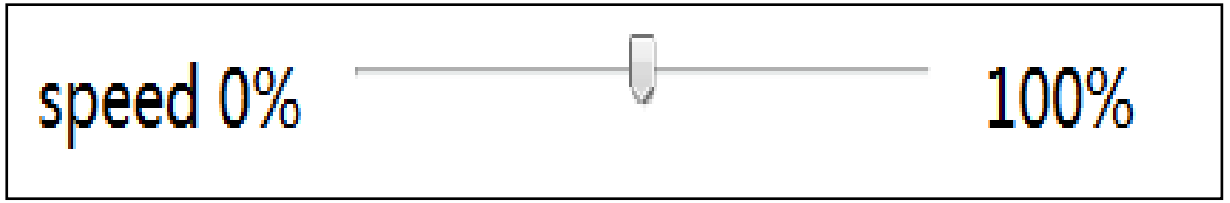
3.Using keydown Event

```
window.addEventListener("keydown", function() {
  switch (event.keyCode) {//按键:数字键1,2, 3代替菜单选择
    case 49: // ’1’ key
      direction = !direction;
      break;
    case 50: // ’2’ key
      delay /= 2.0;
      break;
    case 51: // ’3’ key
      delay *= 2.0;
      break;
  }
});
```

Don’t Know Unicode

```
window.onkeydown = function(event) {
  var key = String.fromCharCode(event.keyCode);
  switch (key) {按键:数字键1, 2, 3代替菜单选择
    case ’1’:
      direction = !direction;
      break;
    case ’2’:
      delay /= 2.0;
      break;
    case ’3’:
      delay *= 2.0;
      break;
  }};
```

4. Slider Element




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

- In HTML file: Puts slider on page
  - Give it an identifier
  - Give it minimum and maximum values
  - Give it a step size needed to generate an event
  - Give it an initial value
- Use div tag to put below canvas

onchange Event Listener

下面两种方式都可以定义滑动条的回调函数

```
document.getElementById("slider").onchange =
function(event)
{
        delay= 100-event.target.value;
};
```

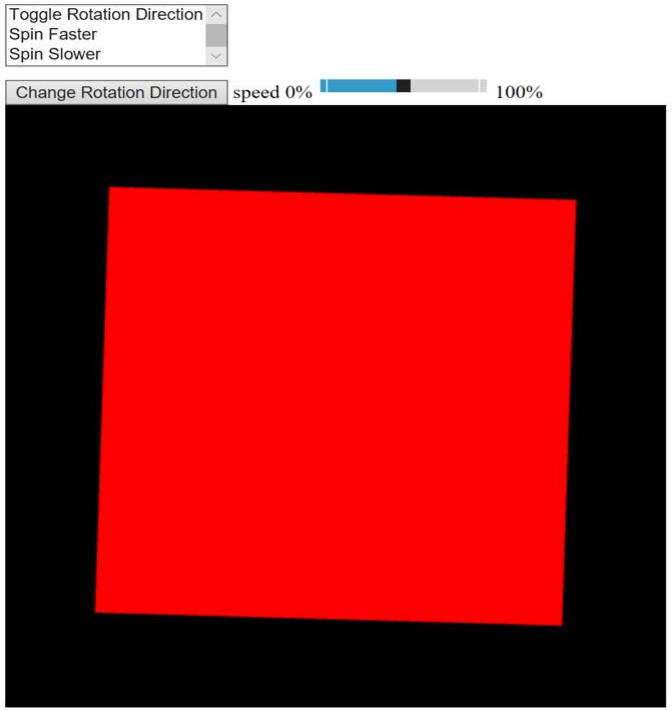
```
document.getElementById("slider").onchange =
  function()
 {
         delay =100- event.srcElement.value;
};
```

Objectives

- Animation
- Event Input Programming
- **Example：Rotate Square（动画与交互）**
  - **Simple but Slow Method~JS Programming**
  - **Better way ~Vertext Shader Programming**
- More input techniques

Rotating Square

- 按一定速度旋转的正方形动画，可交互改变其旋转速度和方向



1.Simple but Slow Method

在应用程序里计算动画每一帧里顶点的位置
程序结构如下：

RotateSquare_RequestAnimFrame
RotateSquare_setTimeOut
RotateSquare

- *Send original vertices to vertex shader（1次）*

- *Compute new vertices by θ in JS（多次）*
- *Send new vertices to vertex shader（多次）*
- *Render recursively（buffer swap）*

Compute new vertices by θ in JS

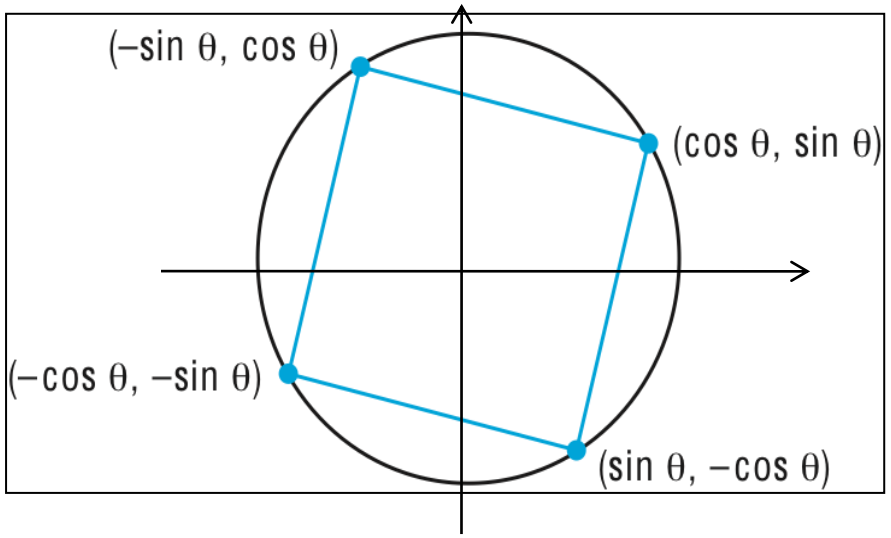
```
var theta = 0.0;
var vertices = [
        vec2(  0,  1 ),  vec2(  -1,  0 ),
        vec2( 1,  0 ),  vec2(  0, -1 )
];
function render() {
    gl.clear( gl.COLOR_BUFFER_BIT );//清屏

    //更新旋转角度，重新计算顶点坐标
    theta += 0.1;
    vertices[0] =vec2(Math.cos(theta),Math.sin(theta));
    vertices[1] =vec2(-Math.sin(theta),Math.cos(theta));
    vertices[2] =vec2(-Math.cos(theta),-Math.sin(theta));
    vertices[3] =vec2(Math.sin(theta),-Math.cos(theta));

    //发送顶点坐标
    gl.bufferData( gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW );

    //绘制正方形
    gl.drawArrays( gl.TRIANGLE_FAN, 0, 4 );

    //要求浏览器显示下次刷新时将要绘制的内容，双帧，递归调用绘制函数
    window.requestAnimFrame(render);
}
```



Consider the four point, circle radius=1

2.Better Way

顶点着色器中编码实现，效率高

- **减少了每次新顶点数据从CPU到GPU的传送**
- **新顶点坐标的计算在GPU上并行执行**

rotatingSquare1
rotatingSquare2
rotatingSquare3

- ***Send original vertices to vertex shader（1次）***

- *Send θ to shader as a uniform variable（多次）*
- *Compute vertices in vertex shader（多次）*
- *Render recursively*

JS: Render Function

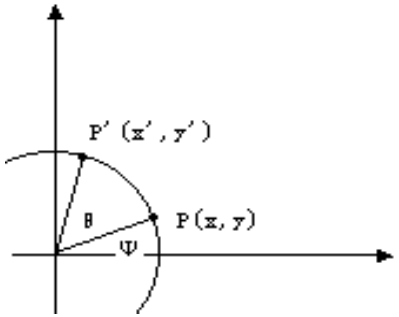
//Send θ to shader as a uniform variable

```
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_FAN, 0, 4);

     window.requestAnimFrame(render);
}
```

Vertex Shader



x'=rcos(θ+Ψ)
=rcosθcosΨ - rsinθsinΨ
y'=rsin(θ+Ψ)
=rsinθcosΨ + rcosθsinΨ
因： $x = r\cos\Psi$ ， $y = r\sin\Psi$
所以： x'= xcosθ- ysinθ
y'= xsinθ+ ycosθ

```
attribute vec4 vPosition;
uniform float theta;//每个顶点都相同的数据
void main()
{//Compute new vertice position
   gl_Position.x = cos(theta) * vPosition.x -sin(theta) * vPosition.y ;
   gl_Position.y = sin(theta) * vPosition.x + cos(theta) * vPosition.y;
   gl_Position.z = 0.0;
   gl_Position.w = 1.0;
}
```

Objectives

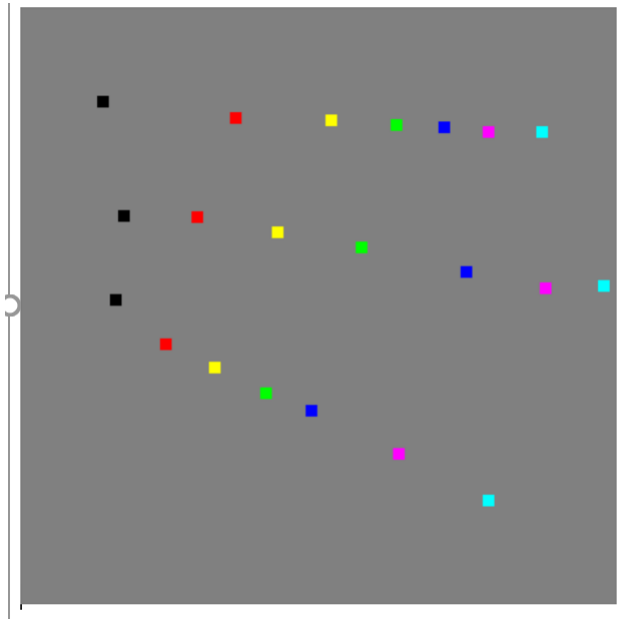
- Animation
- Event Input Programming
- **Example: Rotate Square**
- **More input techniques**
  - **Use the mouse to give locations**
  - **CAD-like Examples**

5.Mouse（3.7 位置输入：mouse click）

- Learn to use the mouse to give locations
  - Must convert from position on canvas to position in application

square.html, square.js:

puts a colored square at location of each mouse click



初始化：可绘制点数，可用的8种颜色

```
var maxNumVertices=600;

var index = 0;

var colors = [
    vec4( 0.0, 0.0, 0.0, 1.0 ),  // black
    vec4( 1.0, 0.0, 0.0, 1.0 ),  // red
    vec4( 1.0, 1.0, 0.0, 1.0 ),  // yellow
    vec4( 0.0, 1.0, 0.0, 1.0 ),  // green
    vec4( 0.0, 0.0, 1.0, 1.0 ),  // blue
    vec4( 1.0, 0.0, 1.0, 1.0 ),  // magenta
    vec4( 0.0, 1.0, 1.0, 1.0 )   // cyan
];
```

Init()函数中：设顶点属性数组（位置，颜色）

注意：需要创建两个属性数据缓存: vBuffer， cBuffer；大小都是maxNumVertices

```
var vBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer);
gl.bufferData( gl.ARRAY_BUFFER, 8*maxNumVertices, gl.STATIC_DRAW );
//gl.bufferData( gl.ARRAY_BUFFER, sizeof['vec2']*maxNumVertices, gl.STATIC_DRAW );

var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);

var cBuffer = gl.createBuffer();
gl.bindBuffer( gl.ARRAY_BUFFER, cBuffer );
gl.bufferData( gl.ARRAY_BUFFER, 16*maxNumVertices, gl.STATIC_DRAW );
//gl.bufferData( gl.ARRAY_BUFFER, sizeof['vec4']*maxNumVertices, gl.STATIC_DRAW );

var vColor = gl.getAttribLocation( program, "vColor" );
gl.vertexAttribPointer( vColor, 4, gl.FLOAT, false, 0, 0 );
gl.enableVertexAttribArray( vColor );
```

Init()函数中：添加 鼠标~click事件的监听器程序

注意：界面添加顶点数据后，用gl.bufferSubData()函数发送数据给GPU

```
canvas.addEventListener("click", function(event){
//canvas.addEventListener("mousedown", function(event){
    //切换到顶点缓存 ，将屏幕坐标转换为世界坐标,每个点vec2含两个浮点数，总共2*4=8个字节
    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
    var t = vec2(2*event.clientX/canvas.width-1,2*(canvas.height-event.clientY)/canvas.height-1);
    gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));
    //gl.bufferSubData(gl.ARRAY_BUFFER, sizeof['vec2']*index, t);//这样写不行！


    //切换到颜色缓存，随机选择一种颜色，每个点颜色代码vec4含四个浮点数，总共4*4=16个字节
    gl.bindBuffer(gl.ARRAY_BUFFER, cBuffer);
    t = vec4(colors[(index)%7]);
    gl.bufferSubData(gl.ARRAY_BUFFER, 16*index, flatten(t));
    //gl.bufferSubData(gl.ARRAY_BUFFER, sizeof['vec4']*index, t);//这样写会报错


    index++;
} );
```

如何将点取的屏幕位置转换为输入顶点位置？
Returning Position from Click Event

```
// add a vertex for each click
   canvas.addEventListener("click", function(event) {
        gl.bindBuffer(gl.ARRAY_BUFFER, vBuffer);
        var t = vec2(-1 + 2*event.clientX/canvas.width,
                     -1 + 2*(canvas.height-event.clientY)/canvas.height);
        gl.bufferData( gl.ARRAY_BUFFER, 8*maxNumVertices, gl.STATIC_DRAW );
        index++;
    });
```

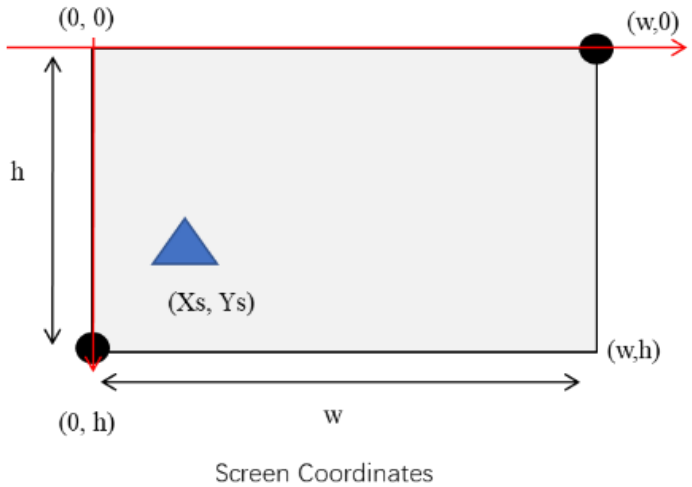
$$Xw = -1 + \frac{2Xs}{w} \qquad Yw = -1 + \frac{2(Ys-h)}{-h}$$

*Canvas specified in HTML file of size canvas.width * canvas.height*

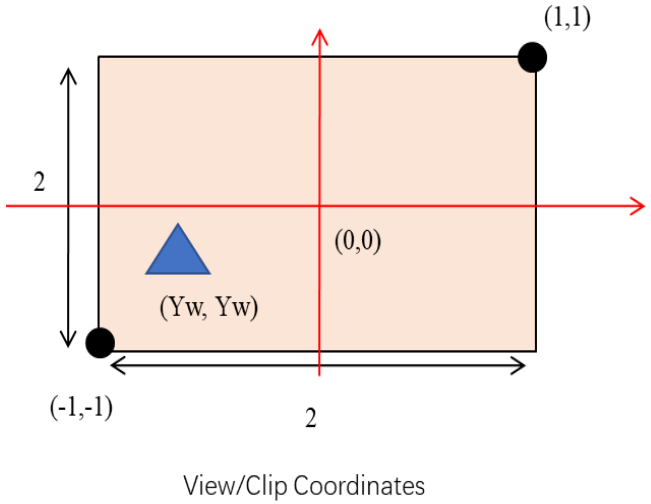
Returned window coordinates are event.clientX and event.clientY

从屏幕坐标到对象坐标(裁剪坐标)

Screen Coordinates



Object/Clip Coordinates



1.直接根据对应成比例进行推导：

$$\frac{Xw-Xwleft}{Xwright-Xwleft}=\frac{Xs-Xsleft}{Xsriht-Xsleft}$$

$$\frac{Yw-Ywbottom}{Ywtop-Ywbottom}=\frac{Ys-Ysbottom}{Ystop-Ysbottom}$$

屏幕视区左上角（xsleft，ysbottom）=（0，0）代入数据：

$$\frac{Xw-(-1)}{1-(-1)}=\frac{Xs-0}{w-0}$$ 得到 $Xw=-1+\frac{2Xs}{w}$

$$\frac{Yw-(-1)}{1-(-1)}=\frac{Ys-h}{0-h}$$ 得到 $Yw=-1+\frac{2(Ys-h)}{-h}=-1+\frac{2(h-Ys)}{h}$

Shader程序中：绘制点

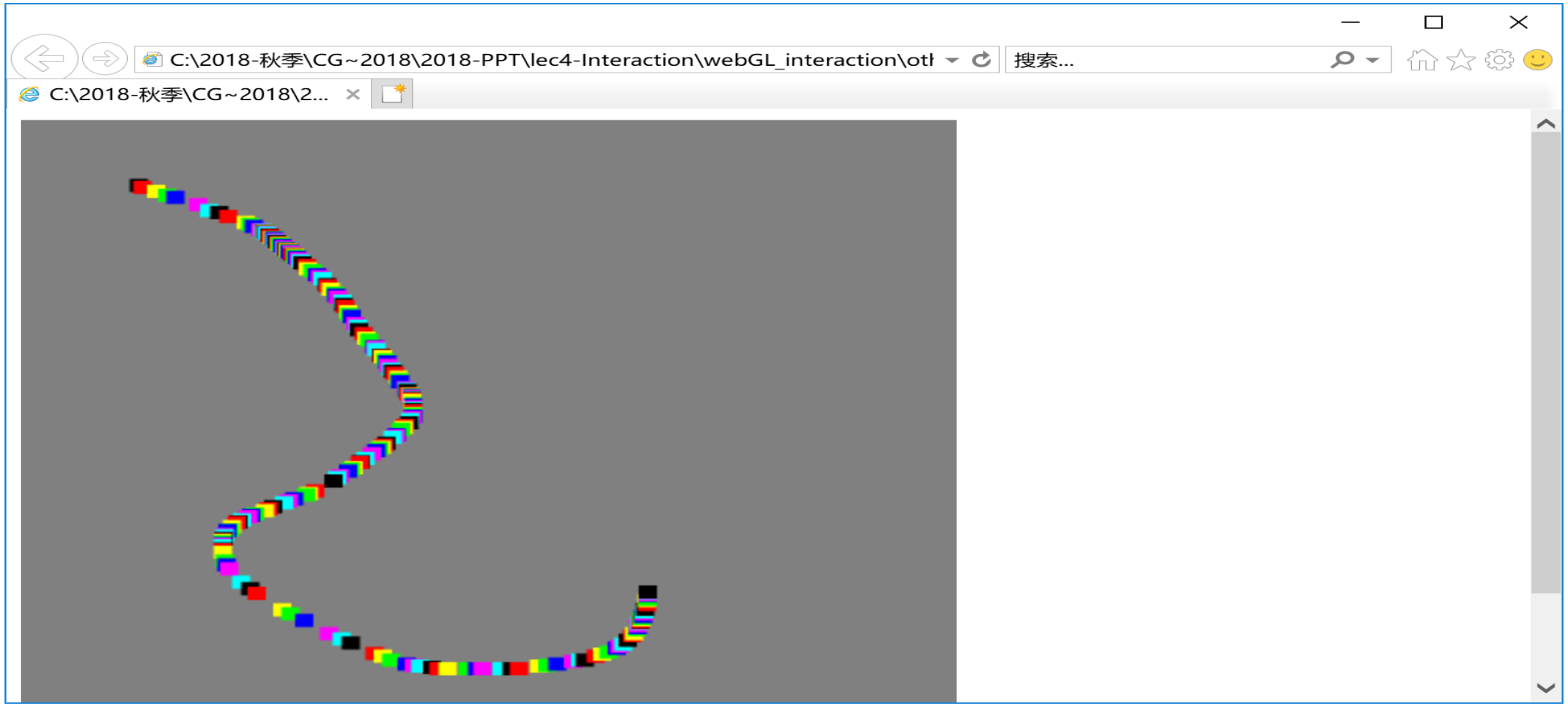
➢顶点着色器接收顶点位置并输出，接受颜色并传递到片元着色器

```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
void main()
{
    gl_Position = vPosition;
    fColor = vColor;
    gl_PointSize = 10.0;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
varying vec4 fColor;
void main()
{
    gl_FragColor = fColor;
}
</script>
```

6.CAD-like Examples：Squarem.html

uses the mousedown event to allow continuous drawing of squares



Init()函数中添加鼠标mousemove事件

注意：这里和Square中鼠标click/mousedown事件区别：

鼠标点下并且移动的时候(mousedown+mousemove)才绘制，

如果鼠标放开(mouseup)就不获取点进行绘制了

```
canvas.addEventListener("mousedown", function(event){
  redraw = true;
});
canvas.addEventListener("mouseup", function(event){
  redraw = false;
});

//canvas.addEventListener("mousedown", function(){
canvas.addEventListener("mousemove", function(event){
    if(redraw) {

    gl.bindBuffer( gl.ARRAY_BUFFER, vBuffer );
    var t = vec2(2*event.clientX/canvas.width-1,
       2*(canvas.height-event.clientY)/canvas.height-1);
    gl.bufferSubData(gl.ARRAY_BUFFER, 8*index, flatten(t));
```

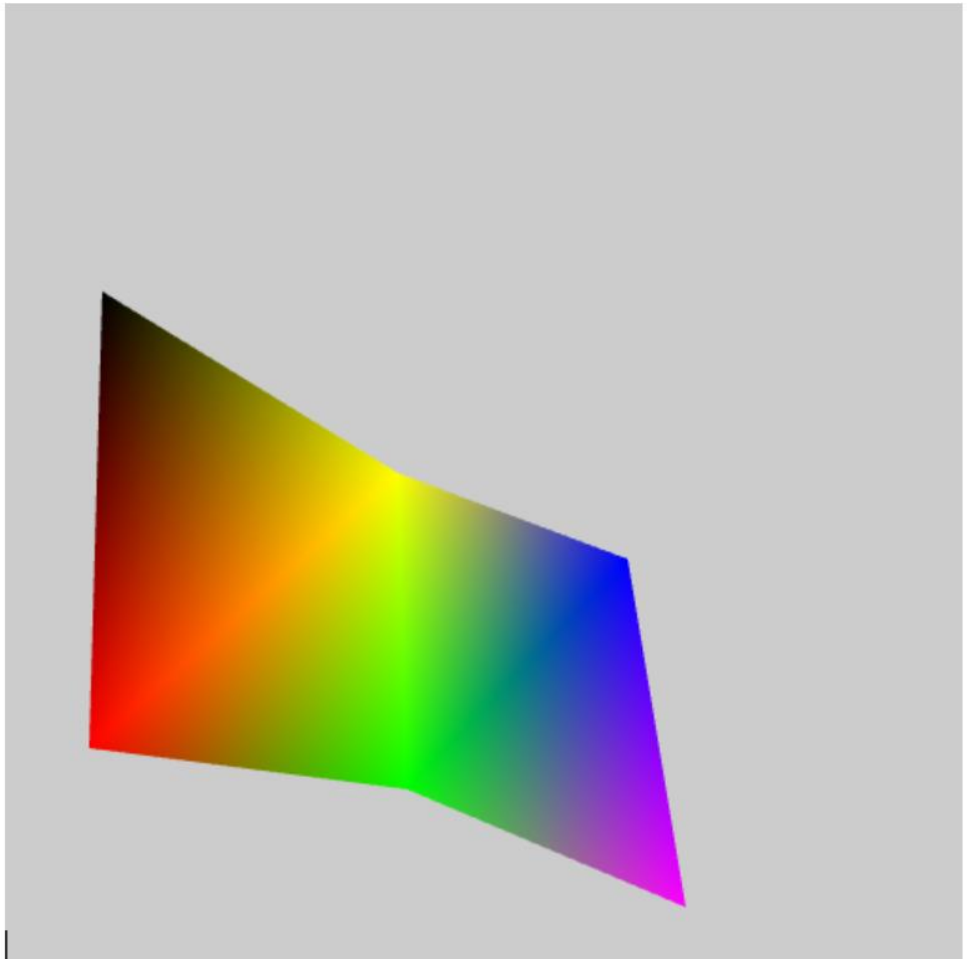
6.CAD-like Examples：triangle

- Each mouse click adds another point to **a triangle strip** at the location of the mouse.

（1，2，3，4，5，6）

=>(1,2.3),(2,3,4)(3,4,5)（4,5,6）

- Shows color interpolation across each triangle.



绘制三角带TRIANGLE_STRIP

```
var maxNumTriangles = 200;
var maxNumVertices  = 3 * maxNumTriangles;
var index = 0;
```

//最多200个三角形

```
function render() {

    gl.clear( gl.COLOR_BUFFER_BIT );

    /*不同于square的地方，
    这里是画三角带:第三个点后开始画三角形，而第四个点开始结合前两个点画三角形。
    每个顶点的颜色从7种颜色中随机选择出来    用varying变量传递插值到片元着色器*/
    gl.drawArrays( gl.TRIANGLE_STRIP, 0, index );

    window.requestAnimFrame(render);
}
```

着色器编程：平滑着色-插值计算-

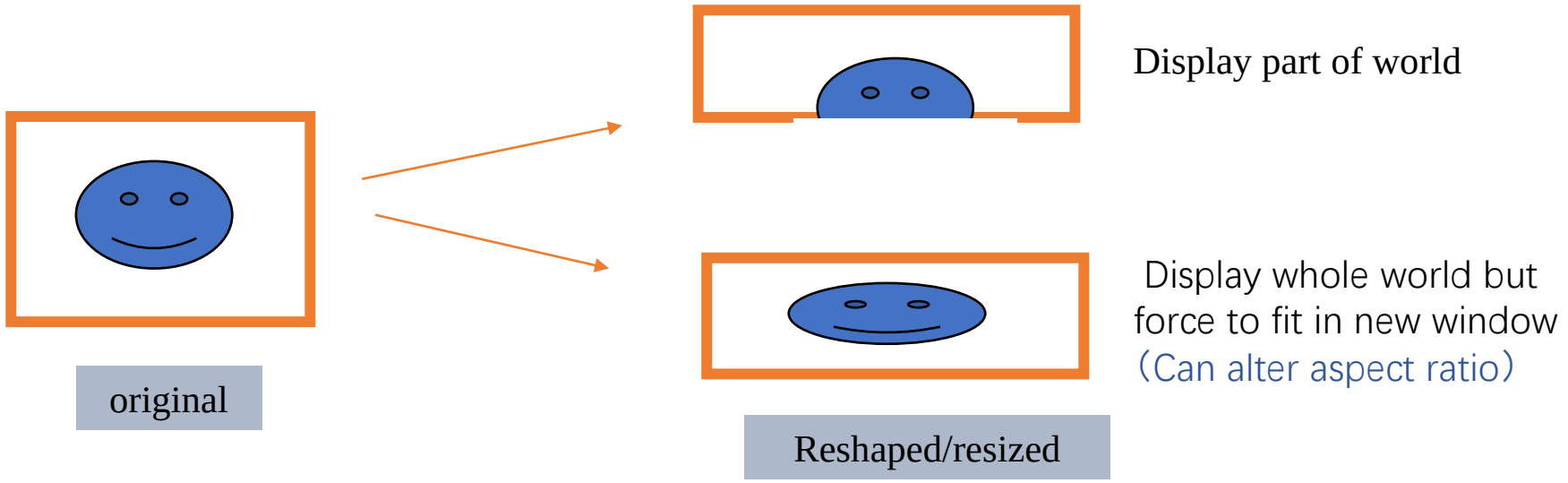
- 因为每个顶点颜色不同，片元颜色有顶点颜色插值计算得到

```
<script id="vertex-shader" type="x-shader/x-vertex">
attribute vec4 vPosition;
attribute vec4 vColor;
varying vec4 fColor;
void main()
{
    gl_Position = vPosition;
    fColor = vColor;
}
</script>

<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;
varying vec4 fColor;
void main()
{
    gl_FragColor = fColor;
}
</script>
```

7.Window Events窗口事件(参3.8 window event)

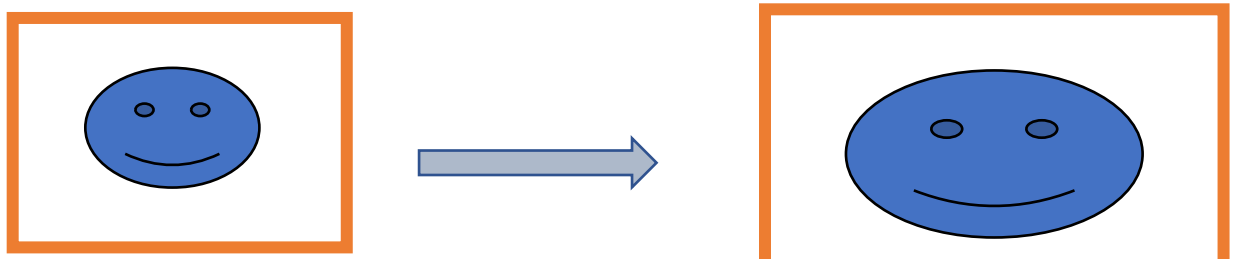
- Events can be generated by actions that affect the canvas window
  - moving or exposing a window(移动，现实窗口)
  - opening a window(打开一个窗口)
  - iconifying/deiconifying a window (图符化一个窗口)
  - resizing a window(改变窗口大小尺寸):以下两种都不当，硬保持比例



窗口大小改变回调函数

参见：作业1代码

```
/* 绘图界面随窗口交互缩放而相应变化，保持比例 */
window.onresize = function(){
    canvas.width = document.body.clientWidth;
    canvas.height = document.body.clientHeight;
    gl.viewport( 0, 0, canvas.width, canvas.height );
 }
```



original

Resized/reshaped

7.CAD-like Examples(cont.)(3.10)

cad1.html: draw a rectangle for each two successive mouse clicks

cad2.html: draws arbitrary polygons

编程作业HW1

1)熟悉程序框架和基本图元绘制方法

在绘图区域用鼠标点击3个以上的点，键盘敲打P键~绘制点图元，敲击L键~绘制线图元，敲打T键盘~绘制多边形图元

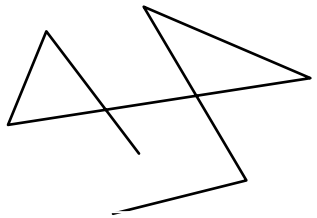
2）了解逐帧动画

在已有逐帧动画程序基础，点“浏览”按钮对话载入关键帧图片可见动画效果。再编程添加滑动条及回调函数实现变化帧频显示动画。

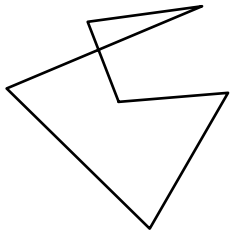
WebGLPrimitives图元

- webGL基本图元：点，线，面

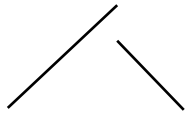
gl.POINTS



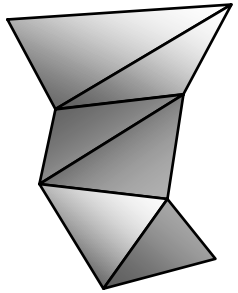
gl.LINE_STRIP



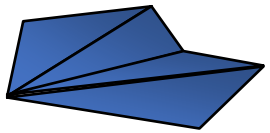
gl.LINE_LOOP



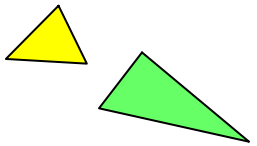
gl.LINES



gl.TRIANGLE_STRIP



gl.TRIANGLE_FAN



gl.TRIANGLES