	Disciplina	CGRAF
	Curso	SI
ALEXANDRE BELETTI FERREIRA	AULA - 15	

Nesta aula trabalharemos com sensores, para tanto, construiremos um primeiro exemplo que permitirá a criação de um objeto e quando o mouse passar sobre ele ter-se-á seu respectivo movimento:

Assim sendo, precisamos do conceito **TouchSensor**:

TouchSensor gera eventos com base nas entradas providas pelo usuário através de um dispositivo de apontamento (geralmente um mouse). Esses eventos indicam se o usuário está apontando para um certo objeto, bem como onde está o ponteiro quando quando o usuário clica o botão do dispositivo e o momento em que ele o fez.

Como para os outros sensores, os objetos monitorados pelo sensor são aqueles que descendem do mesmo nó de agrupamento a que ele está associado (ou seja, seus "nós-irmãos").

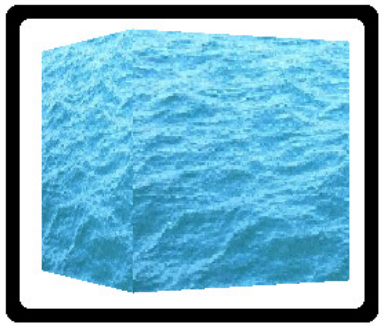
Quando o ponteiro do dispositivo não está sobre nenhum dos objetos mapeados e o usuário o movimenta de maneira que ele aponte para algum dos objetos, o sensor gera um evento *isOver* com valor TRUE. Quando o ponteiro está sobre algum dos objetos e o usuário o movimenta para fora da região mapeada, o sensor envia um evento *isOver* com valor FALSE.

Quando o botão do dispositivo é pressionado e seu ponteiro se encontra sobre algum dos objetos monitorados, o sensor gera um evento *isActive* com valor TRUE. Quando o botão é liberado, o mesmo evento é enviado com valor FALSE.

Quando o usuário clica sobre algum dos objetos mapeados (isto é, pressiona e libera o botão enquanto o ponteiro se encontra sobre algum objeto relevante), um evento *touchTime* é gerado, indicando o horário em que o botão do dispositivo foi liberado. Este evento é muito utilizado para disparar animações, geralmente associado através de uma rota a um evento *set_startTime* de um sensor de tempo (veja a Seção 3.4.4, "*A seqüência de eventos em uma animação*").

```
#VRML V2.0 utf8

Group {
   children [
   # Definindo a criação de um cubo
      DEF Cube Transform {
         children Shape {
            appearance Appearance {
               texture ImageTexture { url "agua.jpg"}
            }
            geometry Box { }
         }
      },
   # Especificando o sensor
      DEF Touch TouchSensor { },
   # Definindo o relogio que controla a animação
      DEF Clock TimeSensor {
         enabled FALSE
         cycleInterval 4.0
         loop TRUE
      },
   # Definindo o caminho da animação
      DEF CubePath OrientationInterpolator {
         key [ 0.0, 0.50, 1.0 ]
         keyValue [
            0.0 1.0 0.0  0.0,
            0.0 1.0 0.0  3.14,
            0.0 1.0 0.0  6.28
         ]
      }
   ]
}
ROUTE Touch.isOver                TO Clock.set_enabled
ROUTE Clock.fraction_changed      TO CubePath.set_fraction
ROUTE CubePath.value_changed      TO Cube.set_rotation
```

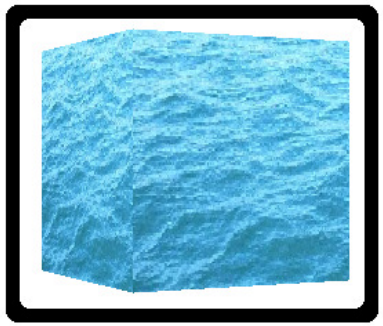


Neste segundo exemplo temos praticamente uma criação equivalente a anterior, porém, o evento do sensor é temporizado num intervalo de tempo pré-determinado e finito.

```
#VRML V2.0 utf8

Group {
   children [
   # Definindo a criação de um cubo
      DEF Cube Transform {
         children Shape {
            appearance Appearance {
               texture ImageTexture {url "agua.jpg"}
            }
            geometry Box { }
         }
      },
   # Especificando o sensor
      DEF Touch TouchSensor { },
```

```
    # Definindo o relogio que controla a animação
        DEF Clock TimeSensor { cycleInterval 4.0 },
    # Definindo o caminho da animação
        DEF CubePath OrientationInterpolator {
            key [ 0.0, 0.50, 1.0 ]
            keyValue [
                0.0 1.0 0.0  0.0,
                0.0 1.0 0.0  3.14,
                0.0 1.0 0.0  6.28
            ]
        }
    ]
}
ROUTE Touch.touchTime              TO Clock.set_startTime
ROUTE Clock.fraction_changed       TO CubePath.set_fraction
ROUTE CubePath.value_changed       TO Cube.set_rotation
```



Agora utilizaremos outro tipo de sensor, a saber: PlaneSensor

Sua função é criar "arrasto" do dispositivo de entrada sobre o(s) objeto(s) mapeado(s).

PlaneSensor mapeia um movimento de arrastar-e-soltar do dispositivo de apontamento (geralmente o mouse), realizado sobre o plano 2D da tela, em uma translação no plano *xy* do sensor, modificando a posição do(s) objeto(s) associados de acordo com o valor especificado em *offset* e a "quantidade" de movimento realizada pelo usuário.

```
#VRML V2.0 utf8

Group {

   children [

      DEF Cube Transform {

         children Shape {

            appearance Appearance {

               texture ImageTexture { url "agua.jpg"}

            }

            geometry Box { }

         }

      },

      DEF Sensor PlaneSensor { }

   ]

}

ROUTE Sensor.translation_changed TO Cube.set_translation
```

Quando utilizamos o PlaneSensor, temos a disposição os seguintes campos:

minPosition	restringe os eventos de translação de maneira que eles estejam acima e à direita deste ponto no plano *xy*;
maxPosition	restringe os eventos de translação de maneira que eles estejam abaixo e à esquerda deste ponto no plano *xy*;
enabled	ativa (quando vale TRUE) ou desativa (FALSE) o sensor;
offset	indica quantos metros os objetos associados ao sensor são transladados quando o usuário começa a arrastar o mouse sobre eles;
autoOffset	determina se o sensor mantém a posição corrente dos objetos a cada novo movimento de arrastar-e-soltar executado pelo usuário. Se *autoOffset* vale FALSE, os objetos voltam à sua posição original a cada novo movimento que ativa o sensor.

Porém no exemplo a seguir exploraremos apenas os campos minPosition e maxPosition:

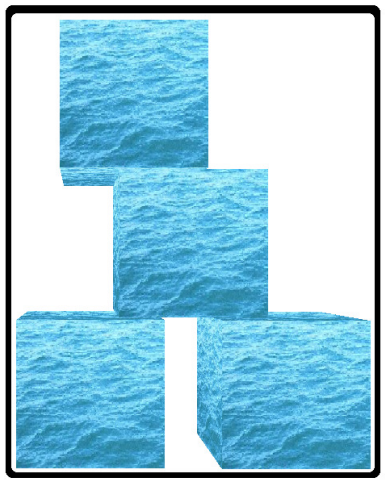
```
#VRML V2.0 utf8
Group {
   children [
      Transform {
         rotation 1.0 0.0 0.0 -1.57
         children DEF Cube Transform {
            children Shape {
               appearance Appearance {
                  texture      ImageTexture { url "agua.jpg"}
               }
               geometry Box { }
            }
         }
      },
```

```
        DEF Sensor PlaneSensor {
            minPosition -2.0 -2.0
            maxPosition  2.0  2.0
        }
    ]
}
ROUTE Sensor.translation_changed TO Cube.set_translation
```

No exemplo a seguir exploraremos o campo off-set numa composição:

```
#VRML V2.0 utf8
Group {
    children [
        Group {
            children [
                DEF Cubo1 Transform {
                    children DEF Padrao Shape {
                        appearance Appearance {
                            texture    ImageTexture { url "agua.jpg"}
                        }
                        geometry Box { }
                    }
                },
                DEF Cubo1Sensor PlaneSensor {
                    offset 0.0 0.0 0.0
                }
            ]
        },
        Group {
            children [
                DEF Cubo2 Transform {
                    translation 2.5 0.0 0.0
                    children USE Padrao
                },

                DEF Cubo2Sensor PlaneSensor {
                    offset 2.5 0.0 0.0
                }
            ]
        },
        Group {
            children [
                DEF Cubo3 Transform {
                    translation 1.5 2.0 0.0
                    children USE Padrao
                },
                DEF Cubo3Sensor PlaneSensor {
                    offset 1.5 2.0 0.0
                }
            ]
        },
        Group {
            children [
                DEF Cubo4 Transform {
                    translation 0.75 4.0 0.0
                    children USE Padrao
                },
                DEF Cubo4Sensor PlaneSensor {
                    offset 0.75 4.0 0.0
                }
            ]
        }
    ]
}
ROUTE Cubo1Sensor.translation_changed TO Cubo1.set_translation
ROUTE Cubo2Sensor.translation_changed TO Cubo2.set_translation
ROUTE Cubo3Sensor.translation_changed TO Cubo3.set_translation
ROUTE Cubo4Sensor.translation_changed TO Cubo4.set_translation
```



Vejamos agora a sensor SphereSensor

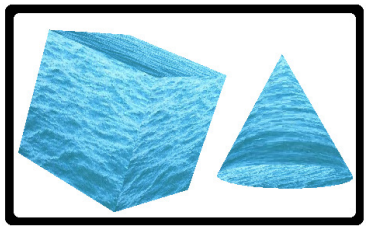
O nó SphereSensor mapeia um movimento de arrastar-e-soltar do dispositivo de apontamento, realizado sobre o plano 2D da tela, em uma rotação no espaço 3D em torno da origem do sistema de coordenadas local. Quando o usuário pressiona o botão do dispositivo enquanto aponta para qualquer dos objetos mapeados pelo sensor, o browser gera uma esfera conceitual, com raio dado pela distância entre o ponto clicado e a origem. Os movimentos de arrasto do ponteiro do dispositivo são interpretados como rotações na esfera, como se o usuário estivesse utilizando um "trackball virtual" (uma esfera que roda sempre em torno de um ponto fixo).

Nele temos a disposição os seguintes campos:

enabled	ativa (quando vale TRUE) ou desativa (FALSE) o sensor;
offset	indica quanto os objetos associados ao sensor são rotacionados quando o usuário começa a arrastar o mouse sobre eles;
autoOffset	determina se o sensor mantém a orientação corrente dos objetos a cada novo movimento de arrastar-e-soltar executado pelo usuário. Se *autoOffset* vale FALSE, os objetos voltam à sua orientação original a cada novo movimento que ativa o sensor.

Veja o exemplo a seguir:

```
#VRML V2.0 utf8
Group {
   children [
      Group {
         children [
            DEF Fig1 Transform {
               children Shape {
                  appearance DEF Cor Appearance {
                     texture   ImageTexture { url "agua.jpg"}
                  }
                  geometry Box { }
               }
            },
            DEF Fig1Sensor SphereSensor { }
         ]
      },
      Group {
         children [
            DEF Fig2 Transform {
               translation 2.5 0.0 0.0
               children Shape {
                  appearance USE Cor
                  geometry Cone { }
               }
            },
            DEF Fig2Sensor SphereSensor { }
         ]
      }
   ]
}
ROUTE Fig1Sensor.rotation_changed TO Fig1.set_rotation
ROUTE Fig2Sensor.rotation_changed TO Fig2.set_rotation
```



Finalmente discutiremos o sensor **CylinderSensor**

CylinderSensor mapeia um movimento de arrastar-e-soltar do dispositivo de apontamento (geralmente o mouse), realizado sobre o plano 2D da tela, em uma rotação no espaço 3D em torno do eixo *y*. Este tipo de sensor (que reponde a ações de "arrastar-e-soltar", assim como PlaneSensor e SphereSensor) pode se comportar de duas maneiras diferentes: como um disco conceitual, quando o usuário clica sobre uma região próxima ao eixo *y* imaginário do cilindro, ou como um cilindro conceitual, quando ele clica sobre regiões mais distantes do eixo.

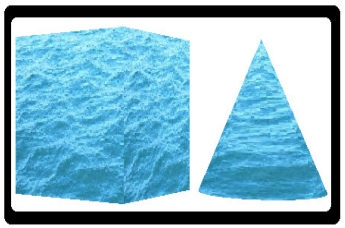
Na prática, o efeito é semelhante ao que aconteceria se dispuséssemos dois objetos sobre um toca-discos: um próximo à borda do disco e outro mais próximo do centro. Como a velocidade angular aumenta em proporção inversa ao raio, o objeto próximo do centro completaria uma volta (360□) mais rápido do que o que está próximo da borda. O mesmo aconte com CylinderSensor: quanto mais próximo do centro do cilindro o usuário clicar, mais rápido os objetos serão rotacionados.

Temos aqui a disposição os seguintes campos:

minAngle	o menor ângulo que um evento *rotation* (evento de rotação para CylinderSensor) pode conter;
maxAngle	o maior ângulo que um evento *rotation* pode conter;
enabled	ativa (quando vale TRUE) ou desativa (FALSE) o sensor;
diskAngle	determina se o sensor se comporta como um cilindro ou como um disco conceituais;
offset	indica quantos radianos os objetos associados ao sensor são rotacionados quando o usuário começa a arrastar o mouse sobre eles (equivale a um ângulo inicial de rotação);
autoOffset	determina se o sensor mantém a orientação corrente dos objetos a cada novo movimento de arrastar-e-soltar executado pelo usuário. Se *autoOffset* vale FALSE, os objetos voltam à sua orientação original a cada novo movimento que ativa o sensor.

Vejamos o exemplo abaixo e sua utilização:

```
#VRML V2.0 utf8
Group {
   children [
      Group {
         children [
            DEF Fig1 Transform {
               children Shape {
                  appearance DEF Cor Appearance {
                     texture   ImageTexture { url "agua.jpg"}
                  }
                  geometry Box { }
               }
            },
            DEF Fig1Sensor CylinderSensor { }
         ]
      },
      Group {
         children [
```



```
            DEF Fig2 Transform {
               translation 2.5 0.0 0.0
               children Shape {
                  appearance USE Cor
                  geometry Cone { }
               }
            },
            DEF Fig2Sensor CylinderSensor { }
         ]
      }
   ]
}
ROUTE Fig1Sensor.rotation_changed TO Fig1.set_rotation
ROUTE Fig2Sensor.rotation_changed TO Fig2.set_rotation
```