

Programando jogos 2D para a nova geração

André Santee – FLISOL 2010. Campo Grande, MS

www.asantee.net

[andre.sante \[arroba\] gmail.com](mailto:andre.sante@gmail.com)

- Este minicurso tem como objetivos apresentar conceitos básicos do desenvolvimento de jogos e introduzir algumas técnicas que serão amplamente utilizadas em jogos 2D da nova geração.

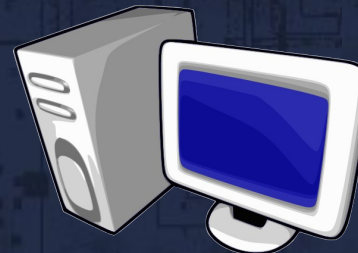
Por que 2D?

- Com o hardware atual há muito espaço para inovações na parte do 2D.
- Mais barato.
- Mais leve.
- Desenvolvimento mais rápido.

Dispositivos de saída e entrada

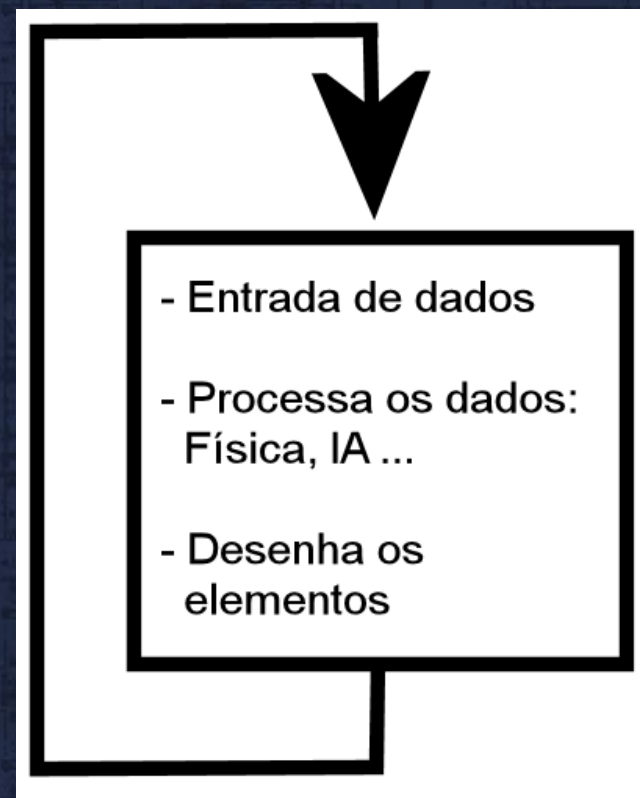
- Saída: monitor/tela
- Entrada: teclado, mouse, joysticks

Para programar videogames precisamos constantemente obter dados dos dispositivos de entrada e exibir os resultados dessas ações através dos dispositivos de saída



Laço principal

- Todo jogo funciona num laço de eventos.
- Cada repetição do laço é um quadro (frame) no jogo.
- Ex.:
O jogador pressionou a barra de espaço?
Se sim, então exibimos a animação de tiro e calculamos quanto de dano isso causou ao inimigo

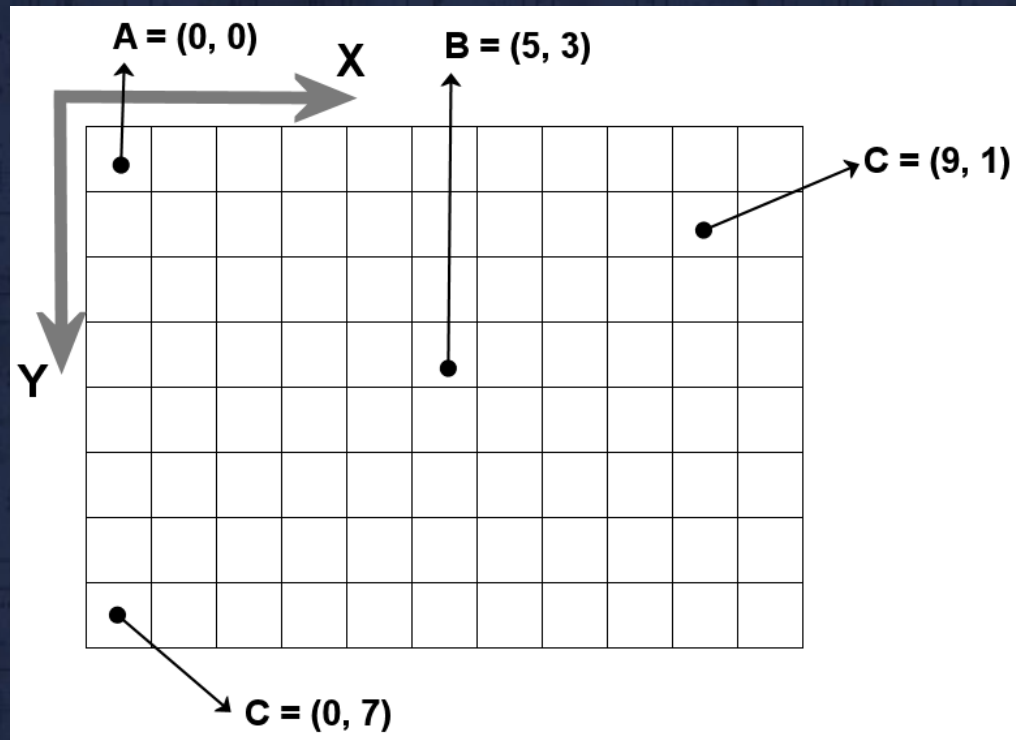


Quadros

Cada quadro corresponde a uma execução do laço principal:



A tela 2D padrão



A direção e a origem dos eixos pode variar de acordo com bibliotecas, API's, plataformas, técnicas de renderização, etc.

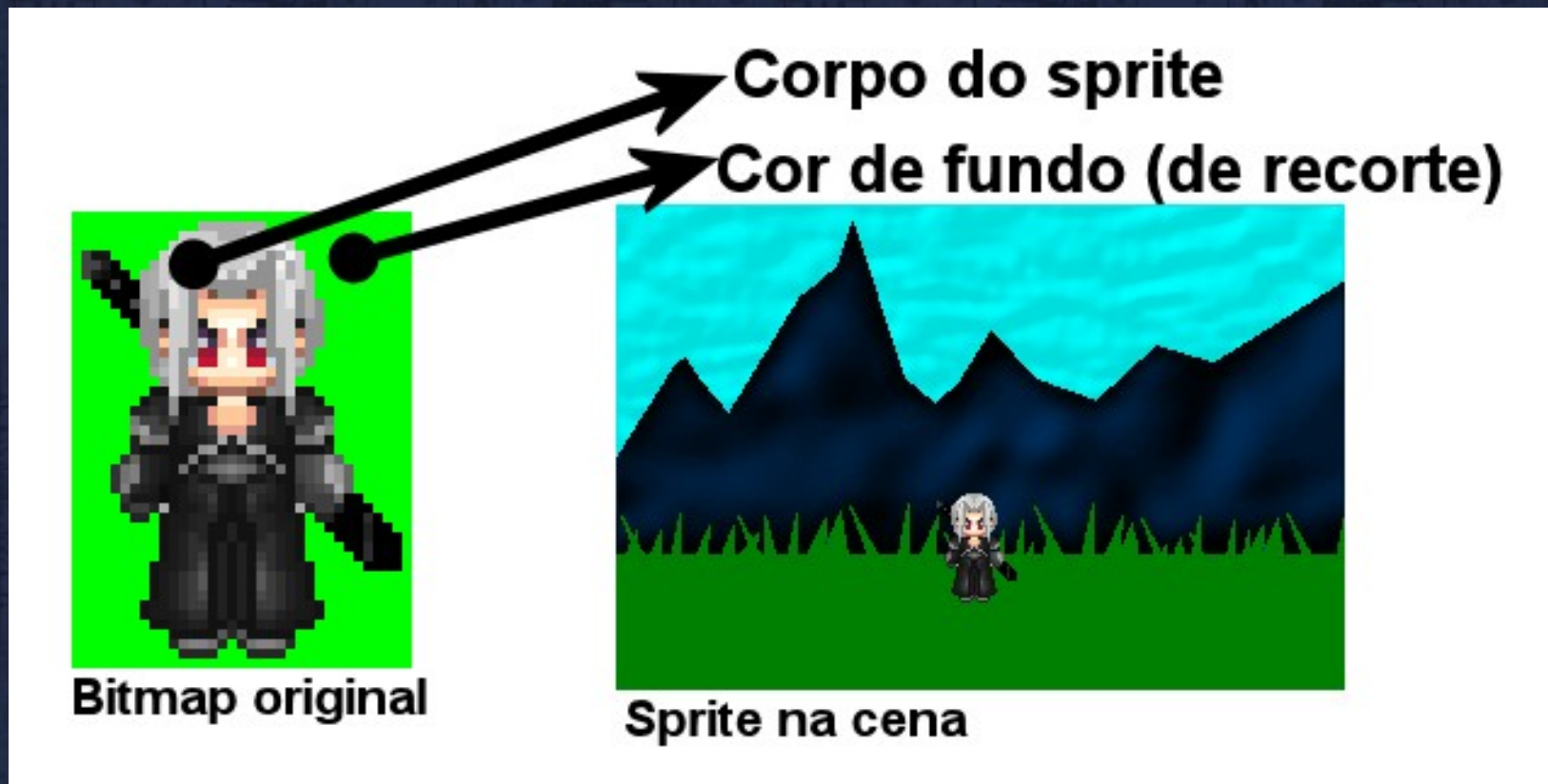
Composição de cenas



Os elementos normalmente precisam ser desenhados de trás para frente

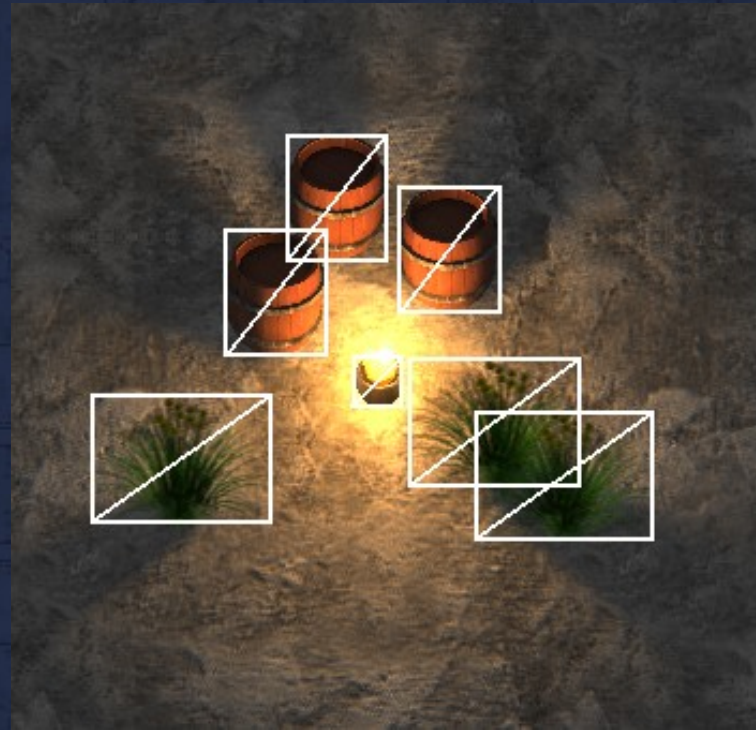
Sprites

Sprites são as "unidades mínimas" da composição de cenas 2D:



Sprites avançados

Os sprites dos jogos 2D atuais são desenhados à partir de polígonos texturizados e renderizados em projeção ortogonal:



Sprites avançados

Os jogos 2D que antes utilizavam somente *tiles* fixos para a composição das cenas, hoje podem ser baseados em *entidades* posicionadas arbitrariamente:





Ethanon Engine

- Motor para desenvolvimento de jogos 2D focados no hardware atual.
- Seu desenvolvimento começou em 2009 e hoje encontra-se na versão 0.7.4 já considerada estável.
- Pode ser utilizado para desenvolver jogos 2D de qualquer estilo. Desde puzzles e side-scrollers até RPG's.
- Possui código-fonte aberto sob licença LGPL (software livre). Site: www.asantee.net/ethanon



Ethanon Engine

- O motor é dividido em 3 partes:
 1. Editor de entidades (entity editor)
 2. Editor de efeitos de partículas (particle FX editor)
 3. Editor de cenas (scene editor).
- Utiliza para a programação do jogo a linguagem de script C-like chamada AngelScript.
- O sistema de script é fundamentalmente baseado em *callbacks*.



Ethanon Engine

Callbacks são funções "infiltradas" no laço do jogo que operam diretamente suas entidades:

```
void ETHCallback_Zumbi(ETHEntity @this)
{
    if (Pegou_Alguem(this))
    {
        Comer_o_Cerebro(this);
    }
    else
    {
        Procurar_Cerebros_Frescos(this);
    }
}
```

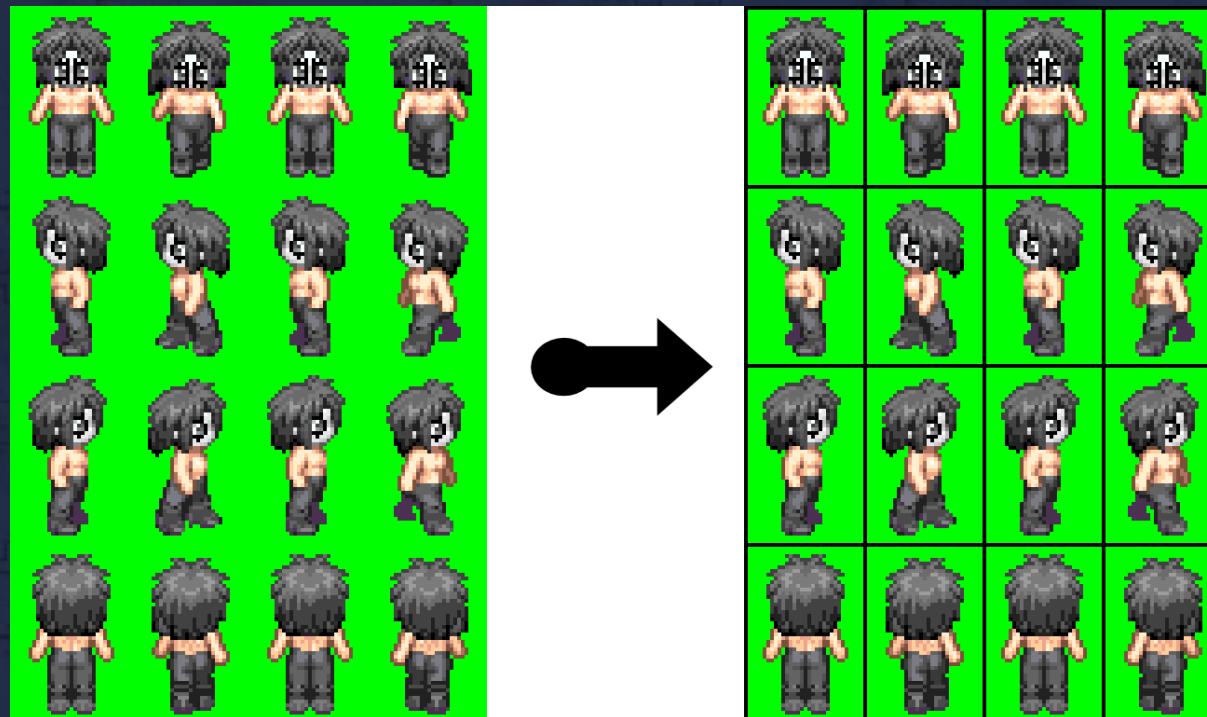
Animação por quadros



A ilusão de animação é conseguida através da combinação de diferentes sprites rapidamente, de modo que a troca dessas imagens cause a impressão de que ela realmente está se movendo.

Animação por quadros

Quando se utiliza sprites animados em um jogo, na maior parte dos casos todos os quadros do sprite ficam agrupados num único bitmap:



Trabalhando com vetores

- Vetores são utilizados para realizar movimentos em 2D.
- Podemos representar vetores (ou pontos) 2D da seguinte maneira:
`navePos = vector2 (20,40)`
- Considere o seguinte sprite:



Trabalhando com vetores

- Considere navePos a variável de vetor/ponto 2D que representa a posição da nave na tela. No momento `navePos = vector2(20,40)`.
- Para movê-la para frente realizamos a seguinte operação:
`navePos = navePos + vector2(20, 0)`



Trabalhando com vetores

- Ao somar `vector(20, 0)` à posição da nave, estamos movendo-a 20 unidades para direita da tela.
- Assim ao desenhá-la no próximo quadro, sua posição será `vector2(40,40)`



NavePos = vector2(40,40)

Animação interativa

- Em jogos eletrônicos a animação é sempre baseada em soma de vetores à posições de personagens (naves, ou seja o que estiver sendo controlado).
- Ao ser atualizada gradativamente a cada quadro (execução do laço principal), temos a impressão de que os objetos estão se movendo na tela.

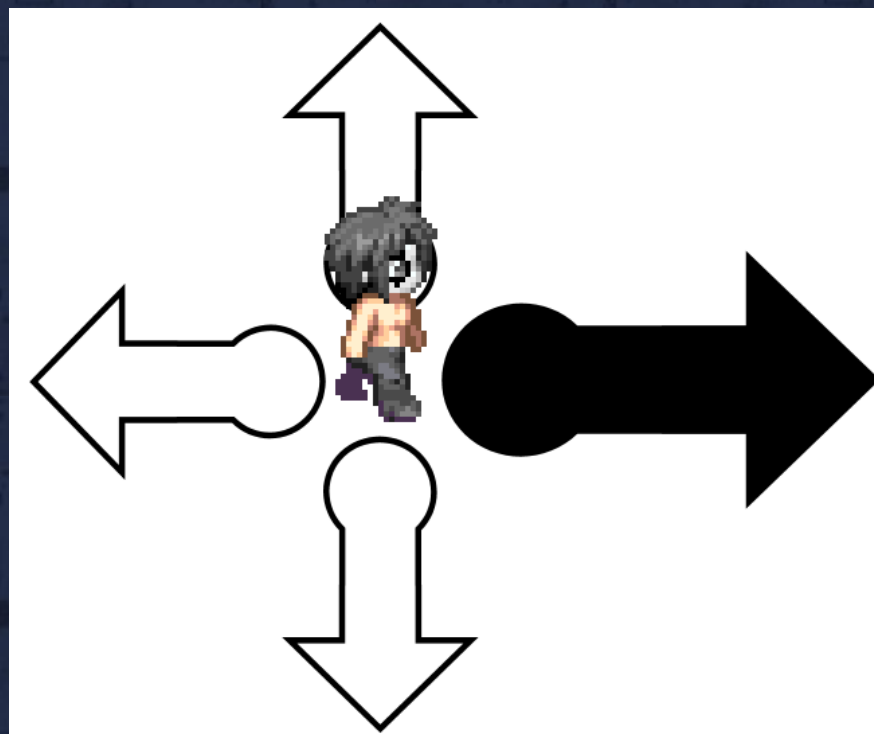
Animação interativa

- O mesmo vale para verificação de dispositivos de entrada:

se pressionar seta direita então:

```
posPersonagem = posPersonagem + vector2(1,0)
```

- Essa verificação, quando feita a cada quadro, permite que o personagem interaja com o jogador.

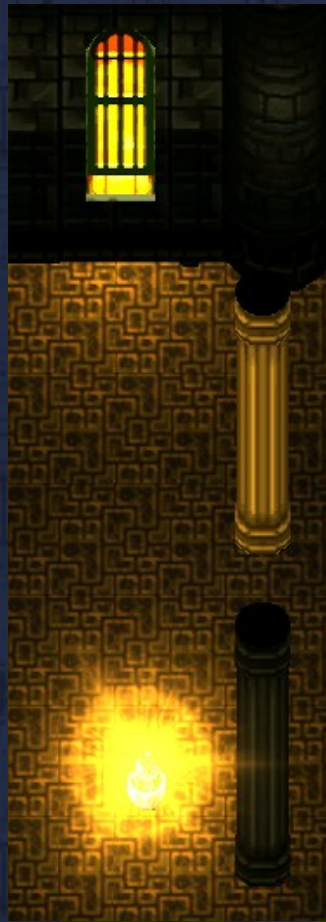


Iluminação em sprites

- O hardware atual permite uso de shaders para efeitos de iluminação 2D mais detalhadas:



Colorização convencional



Somente vertex shaders



Pixel shaders

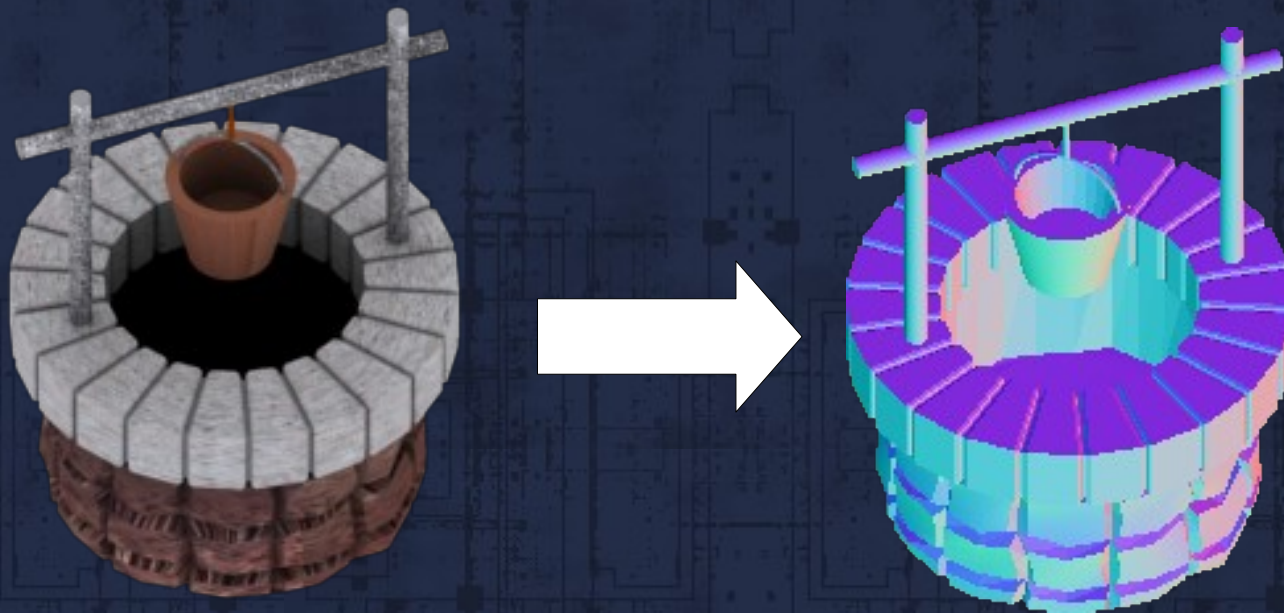
Iluminação por pixel

- A iluminação por pixel em sprite pode ser conseguida ao combinar as cores do *normal map* juntamente à cor original do sprite e as propriedades da fonte de luz:
 - Cada pixel em um normal map corresponde a um vetor tridimensional. Ao invés de utilizar seus valores RGB para cor, o shader de iluminação utiliza-os para eixos X, Y e Z.



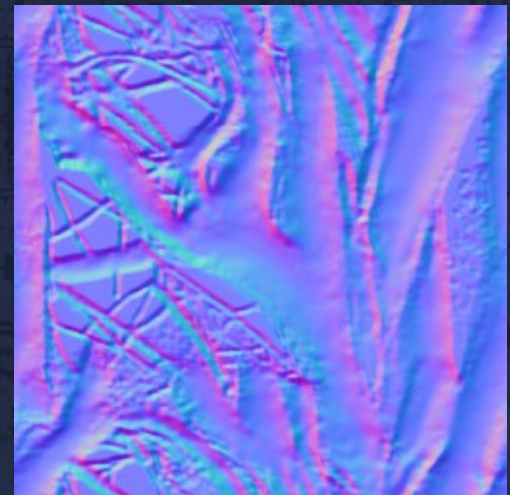
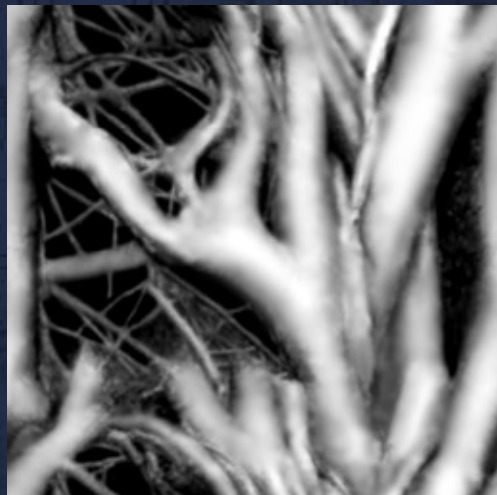
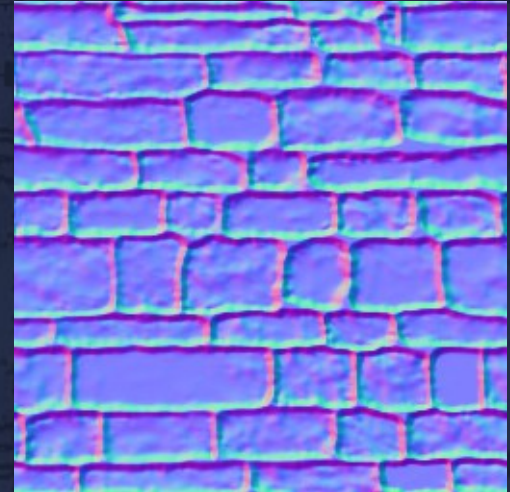
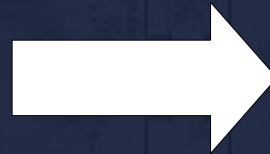
Criando normal maps

- Normal maps podem ser criados à partir de objetos 3D renderizados:



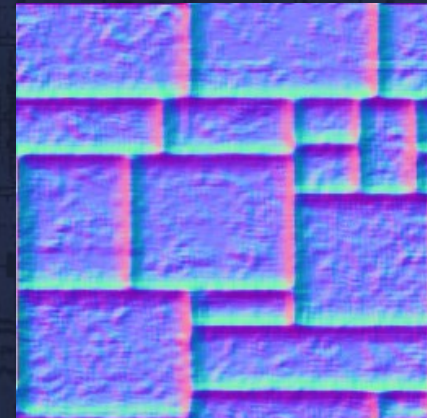
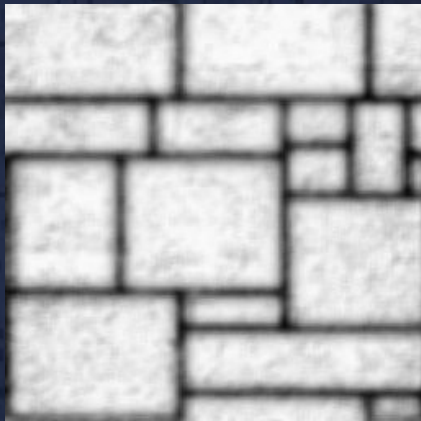
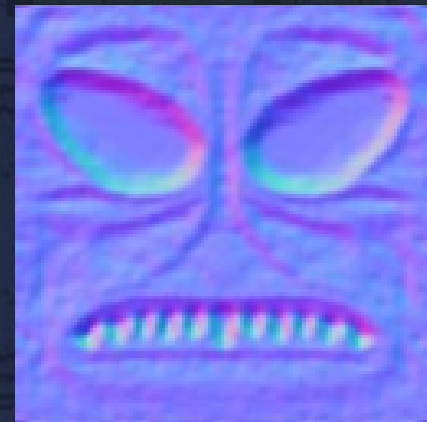
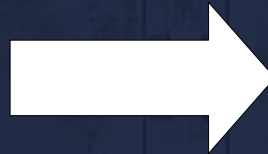
Criando normal maps

- Ou à partir de height maps:



Criando normal maps

- Outros exemplos de height maps:



Criando normal maps

- Muitas ferramentas de edição de imagem possuem plug-ins que transformam height maps em normal maps (ex.: Gimp e Photoshop).
- Existem também outras ferramentas independentes, como a QuickNormal que pode ser baixada em: www.asantee.net

Aplicando normal maps

- Combinando os sprites com seus normal maps, é possível criar cenas com iluminação super detalhada.

