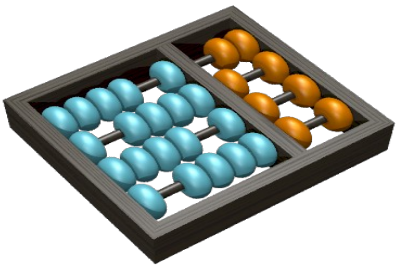
Instituto de Computação
UNICAMP - 2014

Projeto Final de Graduação
MC030



Criação de modelos tridimensionais animados à partir de frames 2D

Bruno Gustavo Salomão Agostini
Orientado por Hélio Pedrini

Introdução

Em computação gráfica, modelagem 3D é o processo de desenvolver uma representação matemática de qualquer superfície tridimensional. Modelos 3D são basicamente uma coleção de pontos no espaço conectados por estruturas geométricas diversas como triângulos, quadrados, entre outras.

Modelos tridimensionais são amplamente utilizados na indústria de jogos e filmes e podem também ser utilizados como moldes de peças e até mesmo órgãos.

O que é o projeto?

Neste projeto o principal objetivo é gerar um modelo tridimensional animado no formato *MD2* a partir de um *Sprite Sheet* no formato *PNG.* Com aplicação direta na área do desenvolvimento de jogos, esta ferramenta possibilita a criação de modelos de qualidade razoável sem que a equipe conte com modeladores profissionais.

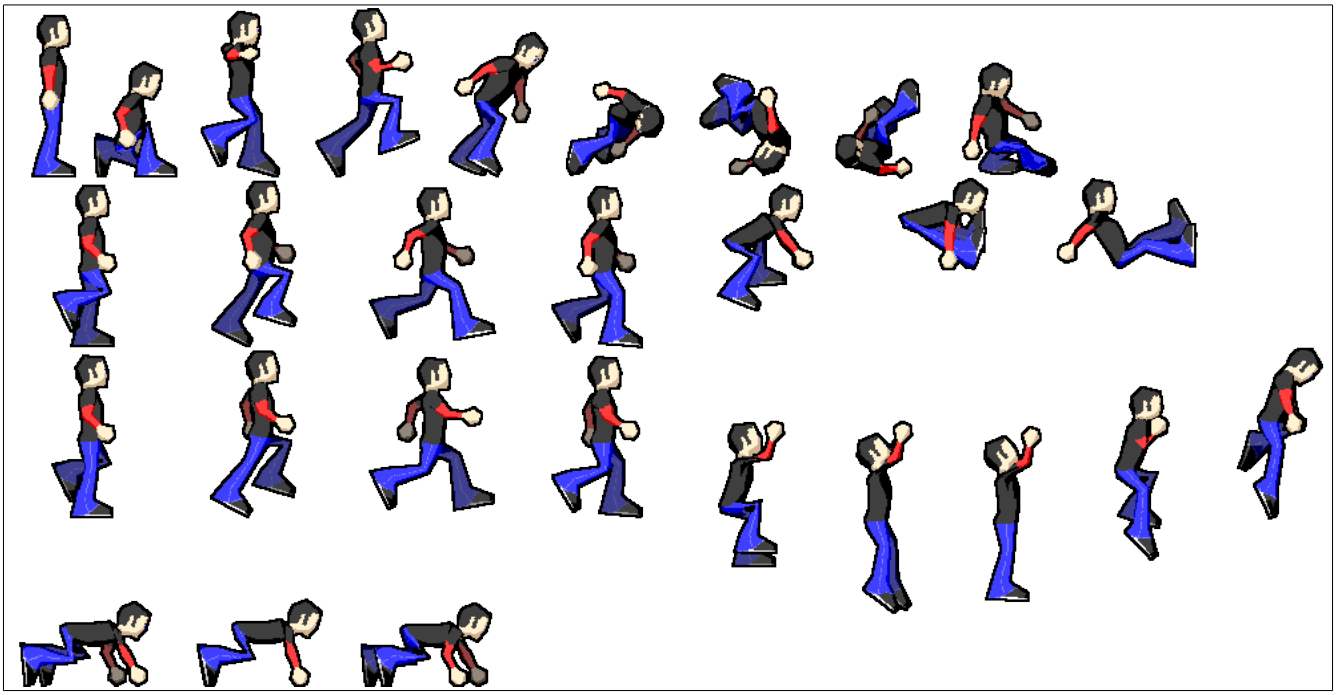
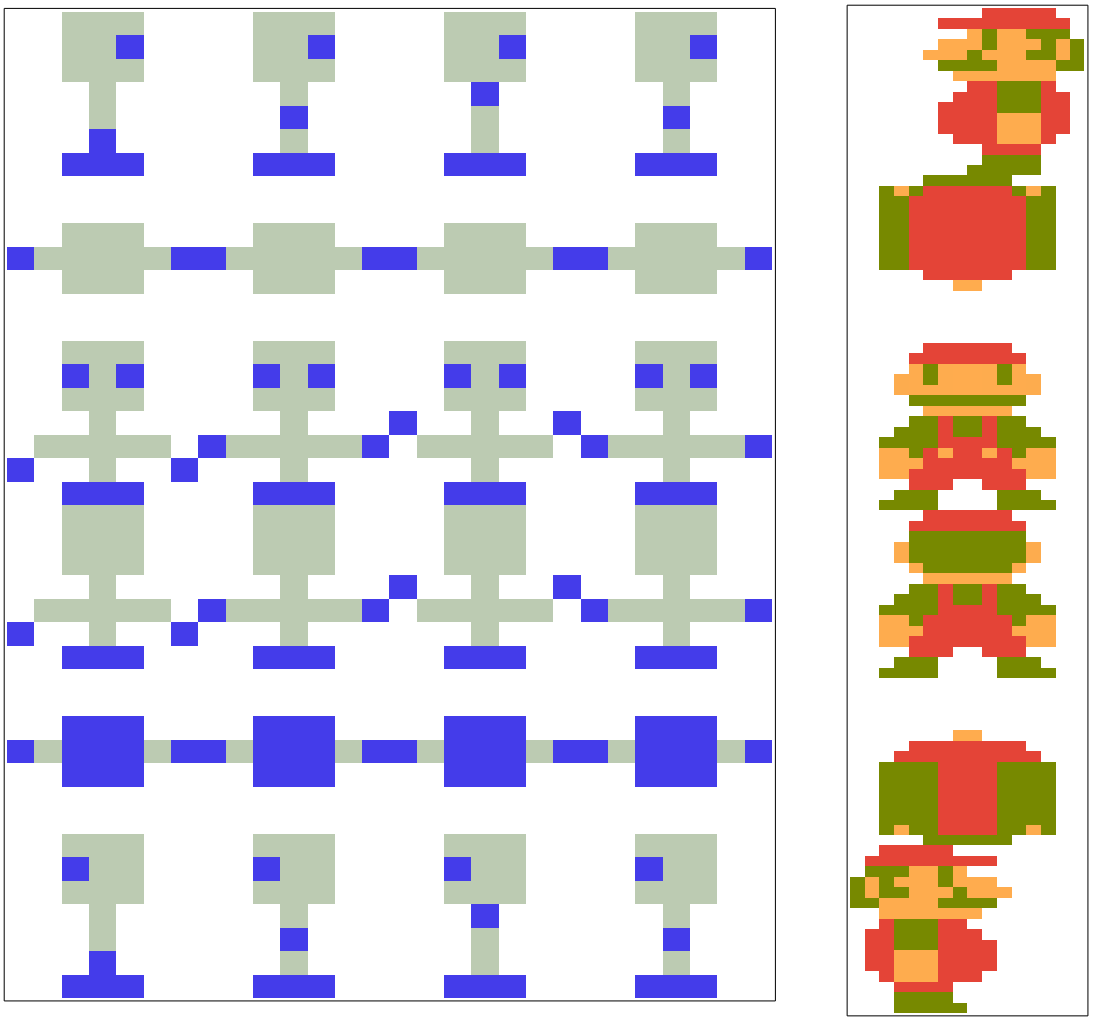


Figura 1: Exemplo de Sprite Sheet convencional, amplamente utilizado em animações e jogos 2D.

Ao invés de utilizar sprites convencionais, o projeto desenvolvido necessita de 6 imagens para cada frame da animação, o que de forma geral costuma ser mais fácil, rápido e barato do que esculpir o modelo tridimensional.



(a) (b)

Figura 2: Sprites aceitos pelo projeto. A imagem (a) fornecerá uma imagem com uma animação de 4 frames. A imagem (b) fornecerá um modelo estático.

Sobre o formato png

O formato *PNG – Portable Network Graphics* – é o formato mais comum utilizado em Sprite Sheets e suporta compressão de dados sem perdas. Após lido, cada pixel é representado por 4 camadas: 3 de cores (*RGB*) e 1 de transparência (camada *Alpha*).

Sobre o formato md2

O formato md2 foi utilizado pela primeira vez no jogo *Quake II*, desenvolvido pela empresa americana *Id Software*. Apesar de antigo, o formato é ainda suportado por inúmeras engines gráficas e de desenvolvimento de jogos, tais qual: *Unity3D, Irrlicht, Ogre*, entre outras.

Esse formato possui um header com algumas informações básicas e um bloco variável de dados, onde armazena vértices, triângulos e texturas. Diferente dos formatos de modelos mais atuais, onde as animações se baseiam em esqueletos, o md2 produz animações através de uma sequência de keyframes estáticos interpolados.

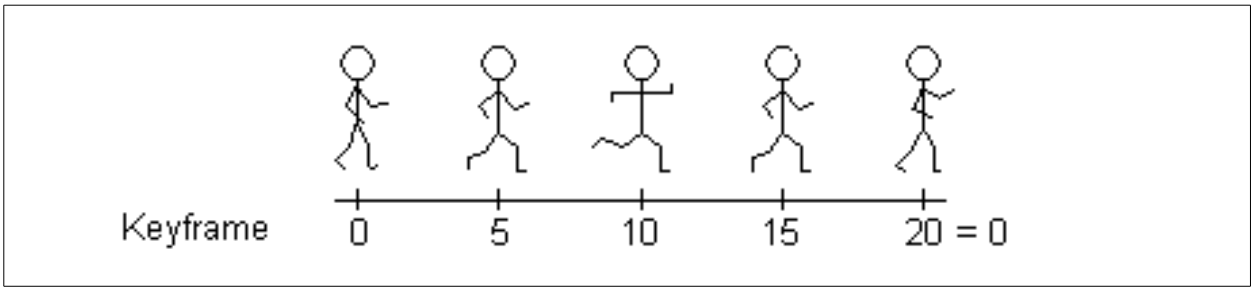


Figura 3: O formato md2 gera animações a partir da interpolação de keyframes.

```
struct Md2Header {
    // If magicNumber != 844121161 or version != 8, so file isn't a MD2
    int magicNumber;
    int version;

    int textureWidth;
    int textureHeight;

    // This will be used to memory allocation
    int frameSizeInBytes;

    int numberOfTextures;
    int numberOfVerticesPerFrame;
    int numberOfTextureCoordinates;
    int numberOfTriangles;
    int numberOfOpenGlCommands;
    int numberOfFrames;

    int offsetToTextureNames;
    int offsetToTextureCoordinates;
    int offsetToTriangles;
    int offsetToFrames;
    int offsetToOpenGlCommands;
    int offsetToEndOfFile;
};
```

Figura 4: MD2 Header.

```
typedef char Skin[64];

struct TextureCoordinates {
    short compressedX;
    short compressedY;
};

struct Triangle {
    short indexToVertices[3];
    short indexToTextures[3];
};

struct Vertex {
    unsigned char compressedCoordinate[3];
    unsigned char indexToLightNormalVector;
};

struct Frame {
    float scaleOfCoordinate[3];
    float translateOfCoordinate[3];

    char name[16];
    Vertex* firstVertexOfThisFrame;
};

struct Md2Data {
    Skin* skins;
    TextureCoordinates* textureCoordinates;
    Triangle* triangles;
    Frame* frames;
    int* glCommandBuffer;
};
```

Figura 5: MD2 Data.

Com base nas estruturas de dados observadas nas figuras 4 e 5, a arquitetura do formato consisti em arquivar uma série de frames para cada modelo, de forma que cada frame conheça seu próprio conjunto de vértices, posteriormente mapeados em triângulos.

Um software que renderiza um arquivo md2 reconhece quais animações aquele modelo possui a partir do nome dos frames. Se um modelo possui 6 frames: *stand00, stand01, walking00, walking01, walking02, walking3;* então isto significa que o modelo possui 2 animações, uma com 2 frames e a outra com 4.

Os *Open GL Commands* são armazenados como inteiros e podem ser utilizados na renderização no intuito de melhorar a performance deste processo.

Como a ferramenta funciona?

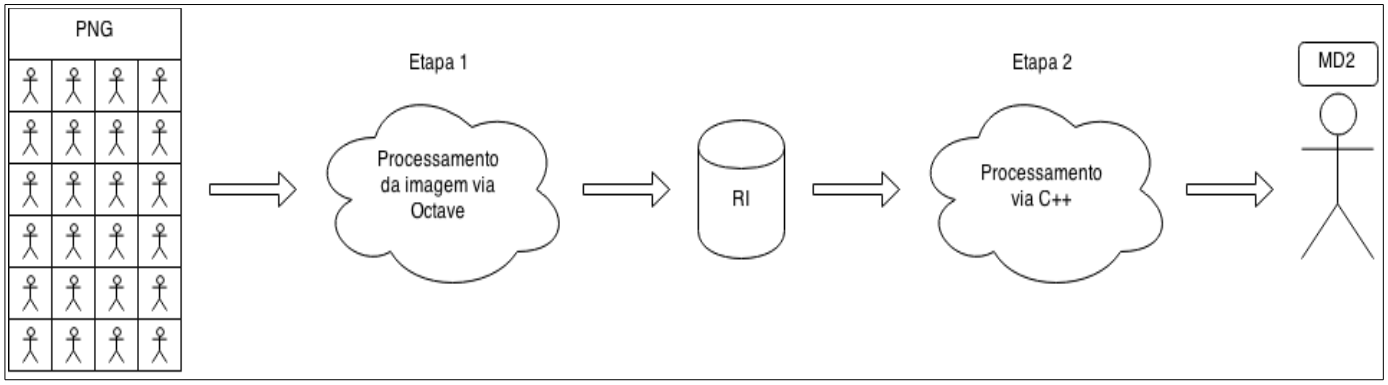


Figura 6: Arquitetura básica da ferramenta.

A ferramenta recebe como entrada um diretório contendo 1 arquivo de configuração e inúmeros sprite sheets que seguem o formato das imagens da figura 2.

O arquivo de configuração, que deve ser nomeado *config*, deve conter 2 linhas. A primeira indica a dimensão, em pixels, que 1 frame possui. Se esse valor for 3, por exemplo, então todos os sprites conterão uma série de frames de dimensão 3x3.

A segunda linha deste arquivo deve conter uma lista com o nome de todas as animações que o modelo final irá conter. Para cada animação existirá 1 sprite no diretório em PNG de mesmo nome. Imagine como exemplo que seu modelo final deva conter 2 animações: *stand* e *walking*. Após ler esses 2 nomes no arquivo de configuração, a ferramenta procuraria por 2 sprites de nome: *stand.png* e *walking.png*.

Com esses dados, a ferramenta gera um arquivo md2 com o mesmo nome do diretório.

Na etapa 1 de processamento, uma série de arquivos temporários são gerados como uma representação intermediária dos dados.

Na etapa 2, esses arquivos temporários são consumidos por uma aplicação desenvolvida em C++ que é responsável pela criação do arquivo md2.

Etapa 1: Fase de Pré-processamento

É nesta etapa que o arquivo de configuração é lido. Para cada animação informada, espera-se um sprite no formato PNG. Com uso da ferramenta livre *Octave*, as imagens são então validadas e processadas. Para cada imagem, armazena-se a quantidade de frames que aquela animação possui e a quantidade total de frames que o modelo deverá conter.
Ainda nesta etapa, são isolados as camadas alphas de todos os frames, que são usados para a criação de uma estrutura intermediária de dados que representa o modelo tridimensional de cada keyframe. Os keyframes são posteriormente agrupados na etapa 2 para formar o arquivo md2.

```
para cada animação:

    procure um sprite no formato PNG;
    verifique que ele possui 6 frames em cada coluna;
    verifique que ele possui uma quantidade inteira de frames em cada linha;

    armazene a quantidade de frames em cada linha em um arquivo;
    atualize o número total de frames do modelo final;

    gere uma representação intermediária do keyframe, esculpindo o sprite;
    armazene em um arquivo essa representação;

armazene o número total de frames em um arquivo;
fim da etapa 1;
```

Figura 7: Pseudocódigo da etapa 1.

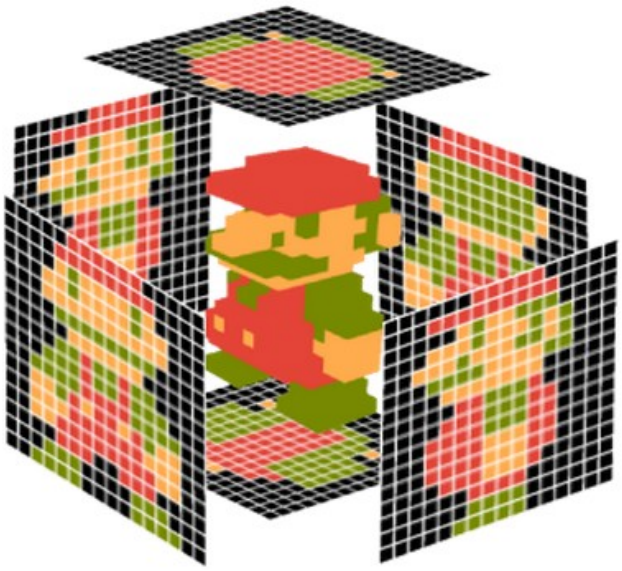


Figura 8: A representação do modelo é obtida esculpindo-se um cubo maciço com os quadros do sprite.

Etapa 2: Gerando o md2

Esta etapa utiliza os dados gerados pela etapa 1 e gera o arquivo md2.

Na final da etapa 1, cada keyframe é representado como uma matriz tridimensional binária, onde o valor 0 indica a existência de um voxel e o valor 1 indica um espaço vazio.

```
para cada keyframe armazenado no fim da etapa 1:
    encontre o arquivo que contém sua representação;

    para cada voxel na matriz tridimensional que o representa:

        se o valor 1 é encontrado:
            armazene um espaço vazio no arquivo md2;

        caso contrário:
            gere 1 cubo visível para representar o voxel;

gere o binário do arquivo md2;
fim da etapa 2;
```

Figura 9: Pseudocódigo da etapa 2.

Exemplos de uso

Um *Shell Script* foi escrito para organizar as etapas da ferramenta. Como argumento recebe um diretório, conforme já descrito anteriormente.

Com os sprites da figura 2, foram gerados 2 modelos com a ferramenta. Os resultados foram então observados auxílio da ferramenta *MM3D*.

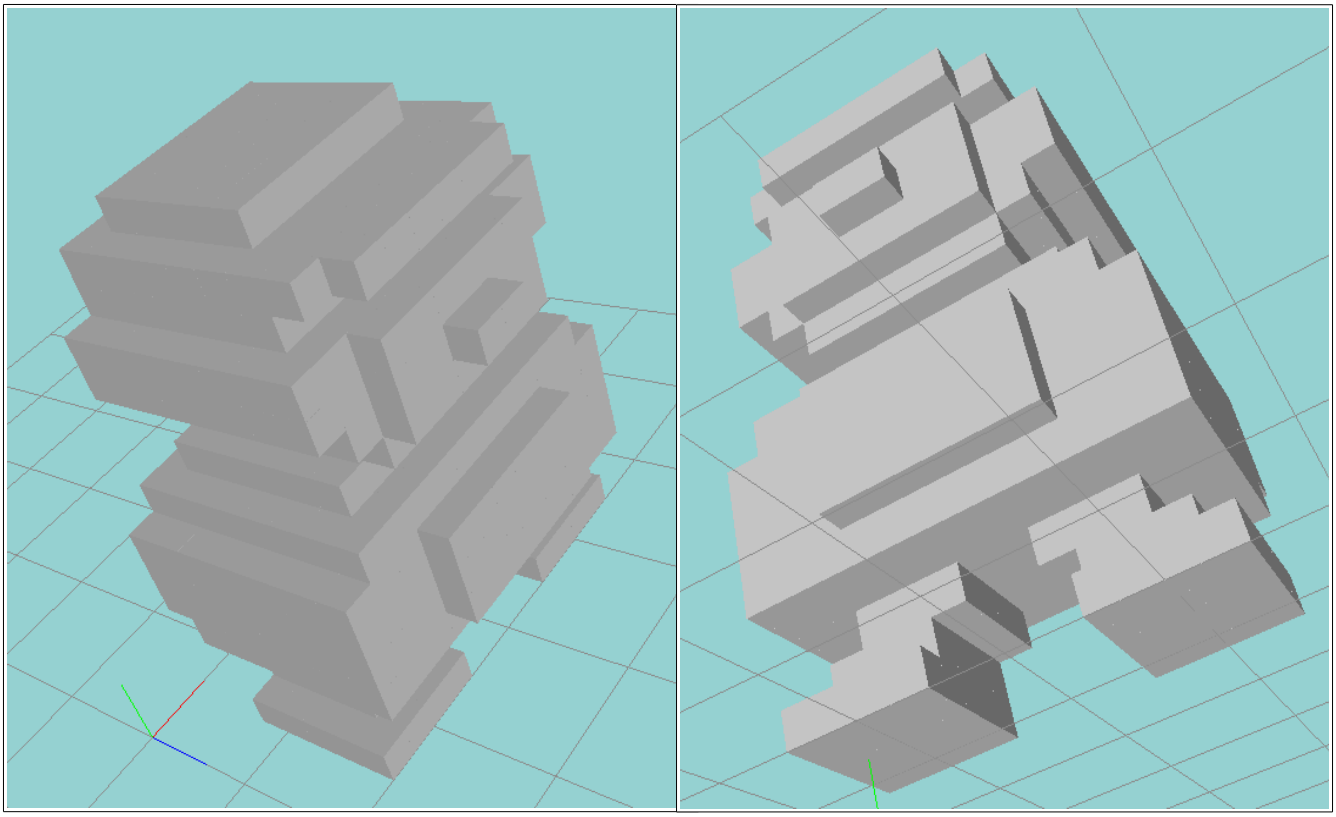


Figura 10: Modelo estático obtido com uso dos sprites da figura 2 (b).

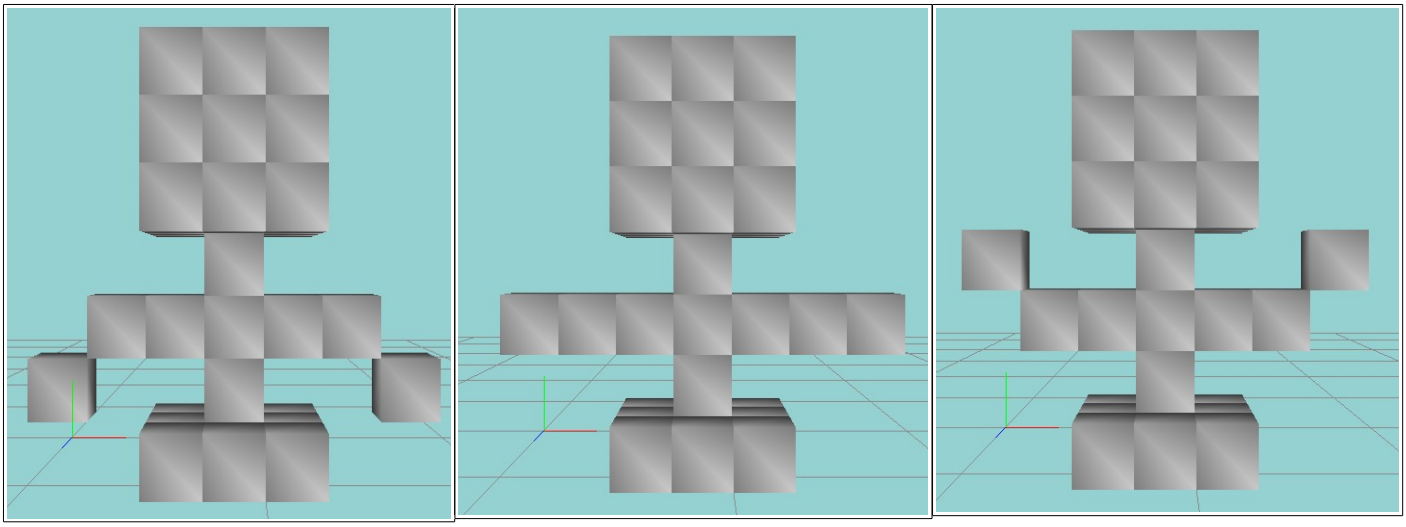


Figura 11: Modelo animado obtido com uso dos sprites da figura 2 (a).

Próximos passos

O intuito deste projeto foi dar o passo inicial para uma ferramenta de uso livre. O link para colaboração é:

https://github.com/brunogsa/spriteToMd2

Como próximos passos, planeja-se:

1. Adição de textura aos modelos gerados;
2. Otimização da quantidade de vértices e triângulos utilizados;
3. Otimização do modelo com uso de Open GL Commands;
4. Criação de uma interface gráfica para os usuários;
5. Expandir o uso para outras plataformas.

Bibliografia

- http://en.wikipedia.org/wiki/3D_modeling
- https://linux.ucla.edu/~phaethon/q3a/formats/md2-schoenblum.html#frame
- http://tfc.duke.free.fr/old/models/md2.htm
- http://tfc.duke.free.fr/coding/md2-specs-en.html
- http://en.wikipedia.org/wiki/MD2_(file_format)
- http://fc07.deviantart.net/fs27/f/2008/083/5/7/Sprite_Sheet_test_by_pfunked.png
- http://en.wikipedia.org/wiki/Portable_Network_Graphics
- http://en.wikipedia.org/wiki/Quake_II_engine
- http://en.wikipedia.org/wiki/Id_Software
- http://unity3d.com
- http://irrlicht.sourceforge.net/
- http://www.ogre3d.org/
- https://github.com/mikolalysenko/mikolalysenko.github.com/tree/master/ShapeCarving
- http://0fps.net/2012/09/18/turning-8-bit-sprites-into-printable-3d-models/
- https://www.gnu.org/software/octave/
- http://www.misfitcode.com/misfitmodel3d/