

Conceitos Básicos

COMPUTAÇÃO GRÁFICA

Freeglut

Vamos utilizar oficialmente o freeglut, o GLUT é uma simplificação da biblioteca OpenGL para o gerenciamento de janelas e afins ser independente de sistema operacional, porém o projeto original está parado desde 1997. O freeglut é uma proposta open source de manter esse projeto vivo.

Criando um projeto no Visual Studio

O primeiro passo é acessar
[Download freeglut](#)

Downloads...

Below are file links for the freeglut project. README files are included. Have fun!

Testing Releases

Feel free to test by downloading a [tarball of current trunk](#), or [grabbing a copy from svn](#), and give us feedback on how it worked for you. All this will eventually become a freeglut 3.4 release.

There are no presently active testing releases.

Stable Releases

[freeglut 3.2.2](#) [*Released: 6 February 2022*]
[freeglut 3.2.1](#) [*Released: 29 September 2019*]
[freeglut 3.2.0](#) [*Released: 16 September 2019*]
[freeglut 3.0.0](#) [*Released: 7 March 2015*]
[freeglut 2.8.1](#) [*Released: 5 April 2013*]
[freeglut 2.8.0](#) [*Released: 2 January 2012*]
[freeglut 2.6.0](#) [*Released: 27 November 2009*]
[freeglut 2.4.0](#) [*Released: 9 June 2005*]
[freeglut 2.2.0](#) [*Released: 12 December 2003*]
[freeglut 2.0.1](#) [*Released: 23 October 2003*]

Prepackaged Releases

The freeglut project does not support packaged versions of freeglut excepting, of course, the tarballs distributed here. However, various members of the community have put time and effort into providing source or binary rollups, and we thank them for their efforts. Here's a list which is likely incomplete:

[Martin Payne's Windows binaries \(MSVC and MinGW\)](#)
[Florian Echtler's MPX Patch](#)

If you have problems with these packages, please contact their maintainers - we of the freeglut team probably can't help.

Development Releases

Criando um projeto no Visual Studio

Como estamos fazendo um projeto no Visual Studio, utilizaremos os binários para Windows do compilador MSVC:
[Binários Windows](#)

Downloads...

Below are file links for the freeglut project. README files are included. Have fun!

Testing Releases

Feel free to test by downloading a [tarball of current trunk](#), or [grabbing a copy from svn](#), and give us feedback on how it worked for you. All this will eventually become a freeglut 3.4 release.

There are no presently active testing releases.

Stable Releases

[freetlut 3.2.2](#) [*Released: 6 February 2022*]
[freetlut 3.2.1](#) [*Released: 29 September 2019*]
[freetlut 3.2.0](#) [*Released: 16 September 2019*]
[freetlut 3.0.0](#) [*Released: 7 March 2015*]
[freetlut 2.8.1](#) [*Released: 5 April 2013*]
[freetlut 2.8.0](#) [*Released: 2 January 2012*]
[freetlut 2.6.0](#) [*Released: 27 November 2009*]
[freetlut 2.4.0](#) [*Released: 9 June 2005*]
[freetlut 2.2.0](#) [*Released: 12 December 2003*]
[freetlut 2.0.1](#) [*Released: 23 October 2003*]

Prepackaged Releases

The freeglut project does not support packaged versions of freeglut excepting, of course, the tarballs distributed here. However, various members of the community have put time and effort into providing source or binary rollups, and we thank them for their efforts. Here's a list which is likely incomplete:

[Martin Payne's Windows binaries \(MSVC and MinGW\)](#)
[Florian Ehtler's MPX Patch](#)



If you have problems with these packages, please contact their maintainers - we of the freeglut team probably can't help.

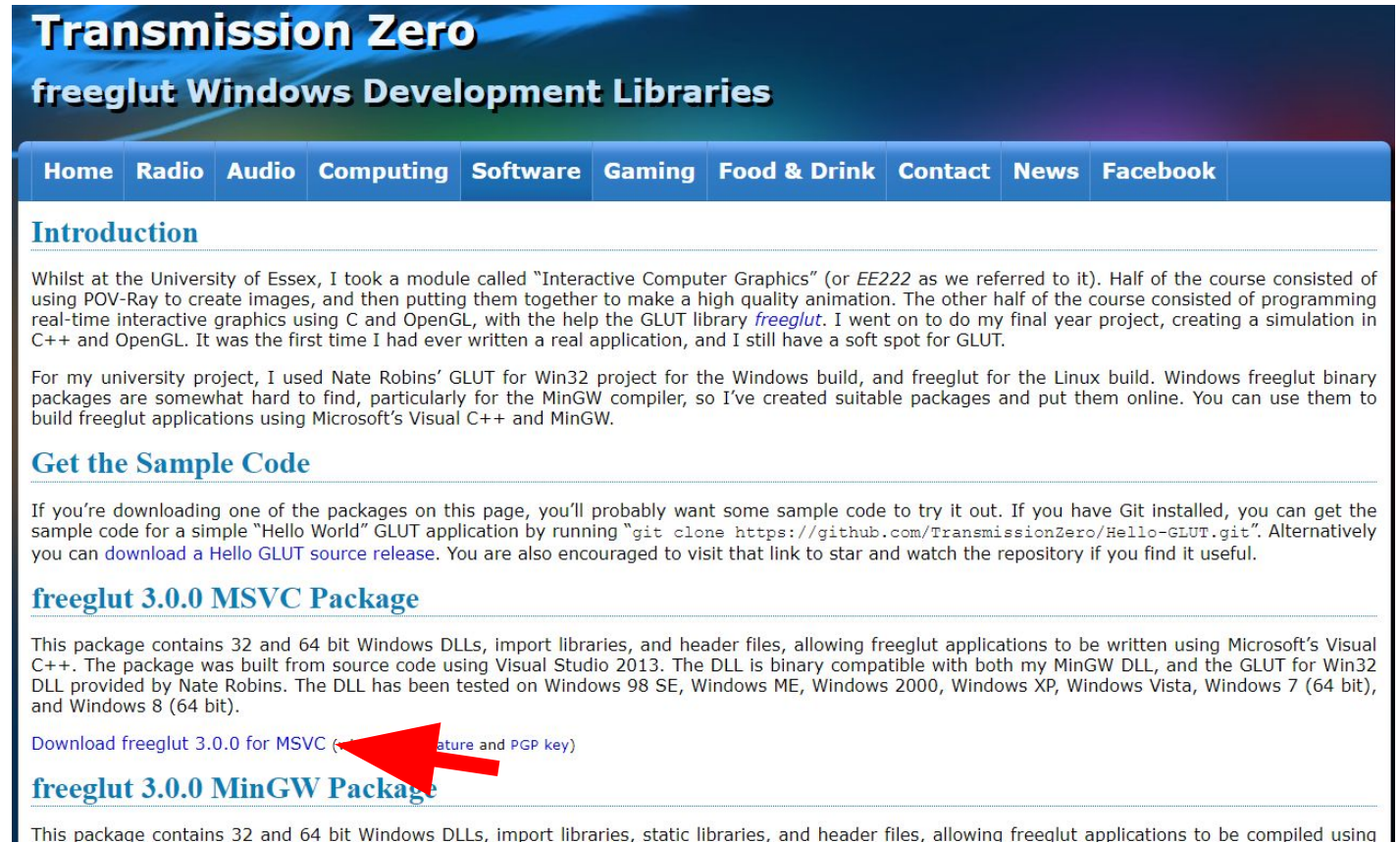
Development Releases

Criando um projeto no Visual Studio

Dentro da página, acessamos os binários específicos para MSVC:

[Binários MSVC](#)

Esse link faz download de um .zip que pode ser descompactado, dentro dele temos a pasta freeglut



Transmission Zero
freeglut Windows Development Libraries

[Home](#) [Radio](#) [Audio](#) [Computing](#) [Software](#) [Gaming](#) [Food & Drink](#) [Contact](#) [News](#) [Facebook](#)

Introduction

Whilst at the University of Essex, I took a module called "Interactive Computer Graphics" (or *EE222* as we referred to it). Half of the course consisted of using POV-Ray to create images, and then putting them together to make a high quality animation. The other half of the course consisted of programming real-time interactive graphics using C and OpenGL, with the help the GLUT library *freeglut*. I went on to do my final year project, creating a simulation in C++ and OpenGL. It was the first time I had ever written a real application, and I still have a soft spot for GLUT.

For my university project, I used Nate Robins' GLUT for Win32 project for the Windows build, and freeglut for the Linux build. Windows freeglut binary packages are somewhat hard to find, particularly for the MinGW compiler, so I've created suitable packages and put them online. You can use them to build freeglut applications using Microsoft's Visual C++ and MinGW.

Get the Sample Code

If you're downloading one of the packages on this page, you'll probably want some sample code to try it out. If you have Git installed, you can get the sample code for a simple "Hello World" GLUT application by running `git clone https://github.com/TransmissionZero/Hello-GLUT.git`. Alternatively you can [download a Hello GLUT source release](#). You are also encouraged to visit that link to star and watch the repository if you find it useful.

freeglut 3.0.0 MSVC Package

This package contains 32 and 64 bit Windows DLLs, import libraries, and header files, allowing freeglut applications to be written using Microsoft's Visual C++. The package was built from source code using Visual Studio 2013. The DLL is binary compatible with both my MinGW DLL, and the GLUT for Win32 DLL provided by Nate Robins. The DLL has been tested on Windows 98 SE, Windows ME, Windows 2000, Windows XP, Windows Vista, Windows 7 (64 bit), and Windows 8 (64 bit).

[Download freeglut 3.0.0 for MSVC \(signature and PGP key\)](#)

freeglut 3.0.0 MinGW Package

This package contains 32 and 64 bit Windows DLLs, import libraries, static libraries, and header files, allowing freeglut applications to be compiled using

Criando um projeto no Visual Studio

Agora vamos fazer a parte do Visual Studio, o primeiro passo é criar um projeto em C++ como Console Application, para isso é necessário adicionar as dependências do C++ no Visual Studio, selecionando elas na instalação ou modificando depois de instalado pelo Installer.

Os computadores da UNIVALI já vêm com as dependências instaladas.

Utilizem o Visual Studio 2019 ou superior para facilitar na criação do projeto.

Criando um projeto no Visual Studio

Agora abram a pasta do projeto, ela pode ser acessada pelo Visual Studio com o botão direito no projeto e “Abrir Pasta no Gerenciador de Arquivos”.

Dentro dessa pasta coloquem a pasta freeglut que estava no .zip do download.

Criando um projeto no Visual Studio

Agora vamos adicionar essa dependência ao projeto, acessando as propriedades (pelo botão direito):

- Primeiro na aba C/C++ -> Geral, em Diretórios de Inclusões Adicionais acrescente:
\$(ProjectDir)freeglut\include
- Depois na aba Vinculador -> Geral, em Diretórios de Bibliotecas Adicionais acrescente:
\$(ProjectDir)freeglut\lib\x64
- Depois na aba Vinculador -> Entrada, em Dependências Adicionais acrescente: ***freeglut.lib***

`$(ProjectDir)` é uma variável que aponta para a pasta do projeto atual, assim podemos modificar a pasta do projeto sem precisar modificar o caminho.

Criando um projeto no Visual Studio

Por último, precisamos fazer uma última ação, ou copiar manualmente a da pasta freeglut/bin/x64 para o projeto, ou colocar um comando para que essa cópia aconteça depois da compilação para não mantermos 2 cópias da dll o tempo todo.

Para isso, na aba Compilar Eventos -> Evento de Pós-Compilação, em linha de comando acrescente: ***copy***

\$(ProjectDir)freeglut\bin\x64\freeglut.dll

Pronto, agora podemos trabalhar com o OpenGL

Programa teste

Baixem o arquivo C1Exemplo.cpp no material didático e copiem o conteúdo para o main.cpp do projeto.

Programa teste

```
glutInitWindowSize (256, 256);  
glutInitWindowPosition (100, 100);
```

Define o tamanho inicial da janela, 256x256 pixels, e a posição inicial do seu canto superior esquerdo na tela, (x, y)=(100, 100). Esses comandos são opcionais.

Programa teste

```
glutCreateWindow ("Desenhando uma linha");
```

Cria uma janela e define seu título como "Desenhando uma linha".

Programa teste

```
glutDisplayFunc(display);
```

Define `display()` como a função de desenho (*display callback*) para a janela corrente. Quando GLUT determina que esta janela deve ser redesenhada, a função de desenho é chamada. A função de desenho deve possuir o seguinte protótipo:

```
void function(void);
```

Programa teste

```
glutKeyboardFunc(keyboard);
```

Indica que sempre que uma tecla for pressionada no teclado, GLUT deverá chamar a função `keyboard()` para tratar eventos de teclado (*keyboard callback*). A função de teclado deve possuir o seguinte protótipo:

```
void function(unsigned char key, int x, int y);
```

Programa teste

```
glutMainLoop();
```

Inicia o *loop* de processamento de desenhos com GLUT. Esta rotina deve ser chamada pelo menos uma vez em um programa que utilize a biblioteca GLUT.

Programa teste

```
glClearColor(1.0, 1.0, 1.0, 1.0);
```

Especifica as intensidade de vermelho (*RED*), verde (*GREEN*) e azul (*BLUE*) utilizadas para limpar a janela. Cada parâmetro pode varia de 0 a 1, o equivalente a uma variação de 0 a 255, usada convencionalmente no sistema de janelas. O último argumento é o canal alfa, usado para compor superfícies transparentes. Como estes conceitos ainda não foram apresentados, o canal alfa deve ser mantido com valor igual a 1.

Programa teste

```
glOrtho (0, 256, 0, 256, -1 ,1);
```

A função glOrtho define as coordenadas do volume de recorte (*clipping volume*), possuindo o seguinte protótipo:

```
void glOrtho()( GLdouble left , GLdouble right , GLdouble bottom ,  
GLdouble top , GLdouble zNear , GLdouble zFar );
```

Programa teste

```
glClear(GL_COLOR_BUFFER_BIT);
```

A função `glClear()` serve para limpar buffers utilizados pelo OpenGL com valores pré-definidos. A máscara utilizada neste exemplo, `(GL_COLOR_BUFFER_BIT)`, diz à função `glClear()` que apenas o buffer de desenho deve ser limpo. Após a execução desta função, a tela ficará branca, uma vez que a `init()` define `(R, G, B)=(1.0, 1.0, 1.0)` como cor de limpeza de tela.

Programa teste

```
glColor3f (0.0, 0.0, 0.0);
```

Especifica (R, G, B)=(0, 0, 0), preto, como a cor de desenho. Todos os objetos desenhados a partir daqui terão cor preta.

Programa teste

```
glBegin(GL_LINES);  
glVertex2i(40,200);  
glVertex2i(200,10);  
glEnd();
```

As funções `glBegin()` e `glEnd()` delimitam os vértices de uma primitiva de desenho ou de um grupo de primitivas. O parâmetro passado para a função especifica o tipo de primitiva a ser desenhado. Neste exemplo, o parâmetro `GL_LINES` indica que os vértices especificados devem ser tratados como pares de pontos que comporão segmentos de reta independentes. A função `glVertex2i()` define as coordenadas de um vértice.

Programa teste

```
void keyboard(unsigned char key, int x, int y){  
    switch (key) {  
        case 27:  
            exit(0);  
            break;  
    }  
}
```

Aqui a função de callback é chamada quando uma tecla é pressionada, nesse caso ele finaliza o programa quando a tecla 27(esc) é pressionada.

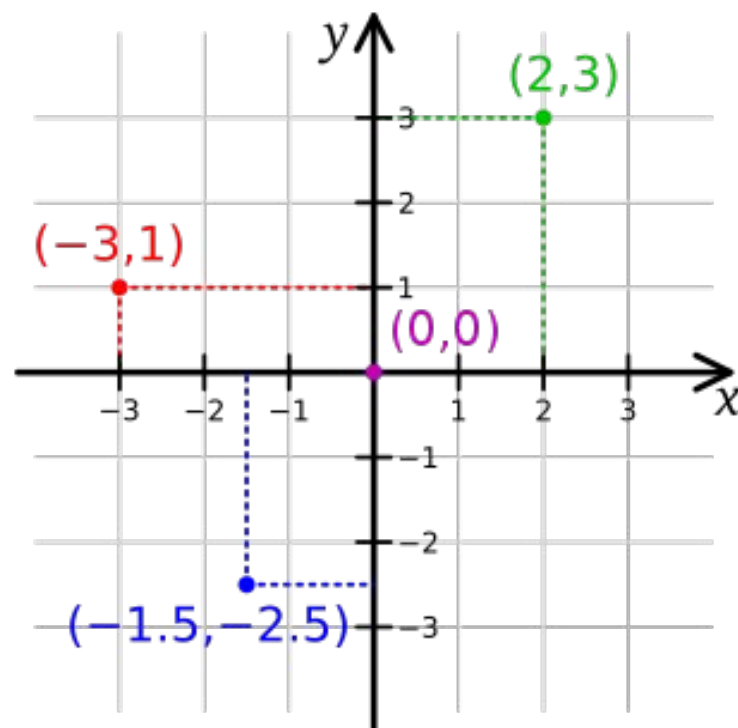
Translação e Rotação

Pensando de forma fácil, se eu quero me movimentar pelo eixo x em 20 unidades de distância:

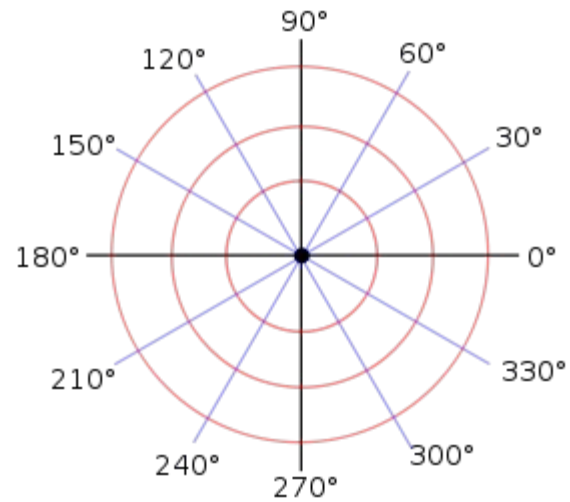
- $X = X + (\text{Deslocamento} * \text{Direção})$

O mesmo vale para o y, mas e se quisermos nos deslocar em uma direção que não seja paralela a um dos eixos, ou pior, rotacionar um objeto, precisamos trabalhar com um pouco de trigonometria básica.

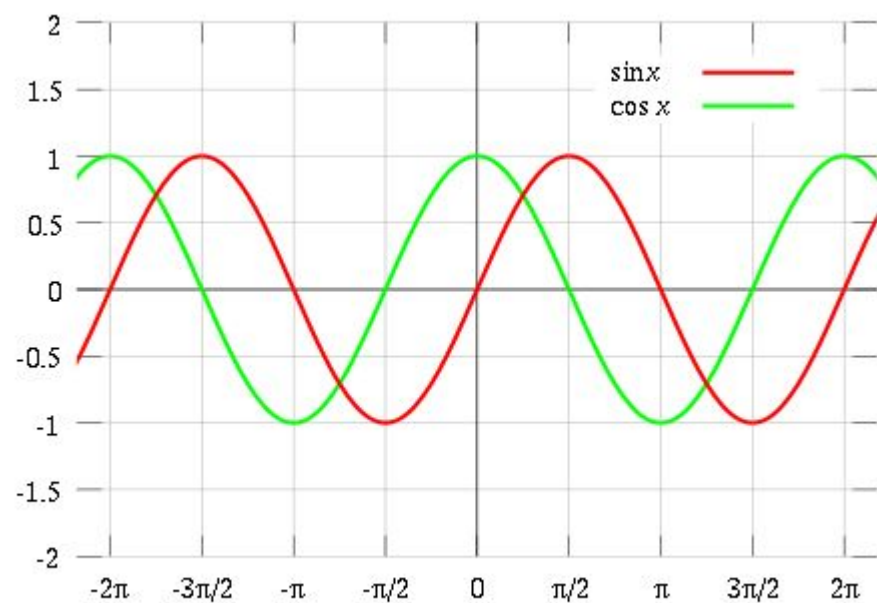
Um pouco de trigonometria



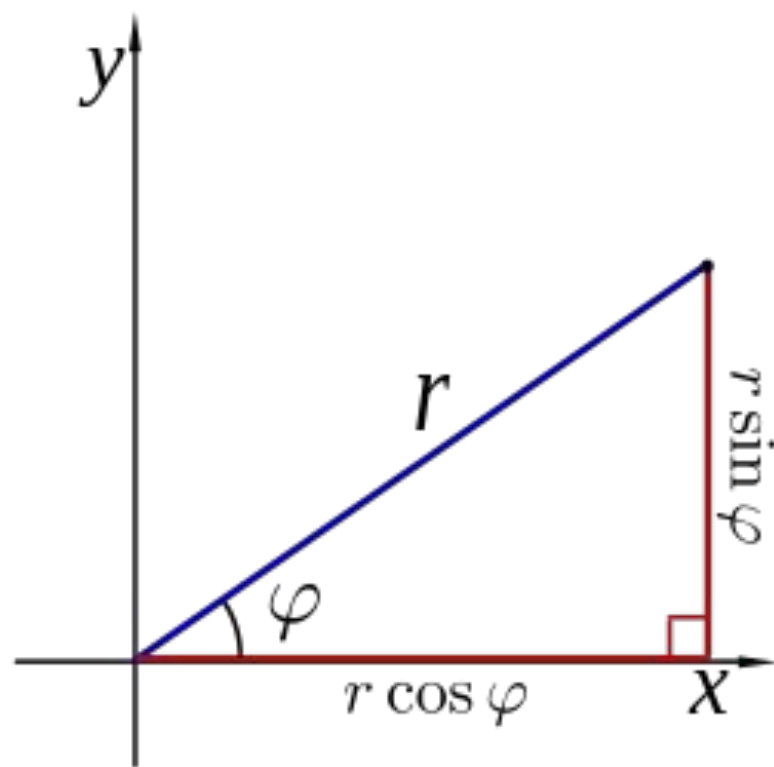
Um pouco de trigonometria



Um pouco de trigonometria



Um pouco de trigonometria

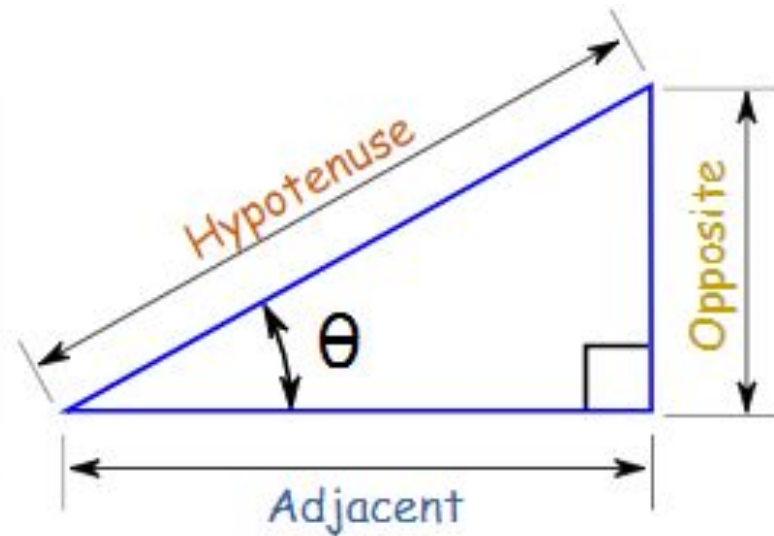


Um pouco de trigonometria

$$\sin \theta = \frac{\text{Opposite}}{\text{Hypotenuse}}$$

$$\cos \theta = \frac{\text{Adjacent}}{\text{Hypotenuse}}$$

$$\tan \theta = \frac{\text{Opposite}}{\text{Adjacent}}$$



Algumas fórmulas

Distância entre dois pontos:

- $\text{Distância} = \sqrt{(\text{XFinal} - \text{XInicial})^2 + (\text{Yfinal} - \text{YInicial})^2}$

Encontrar o ângulo de um ponto a partir da origem do plano cartesiano:

- $\text{Ângulo} = \text{atan2}(\text{Xponto} - \text{Xorigem}, \text{Yponto} - \text{Yorigem})$

Movimentar uma distância por um dado ângulo:

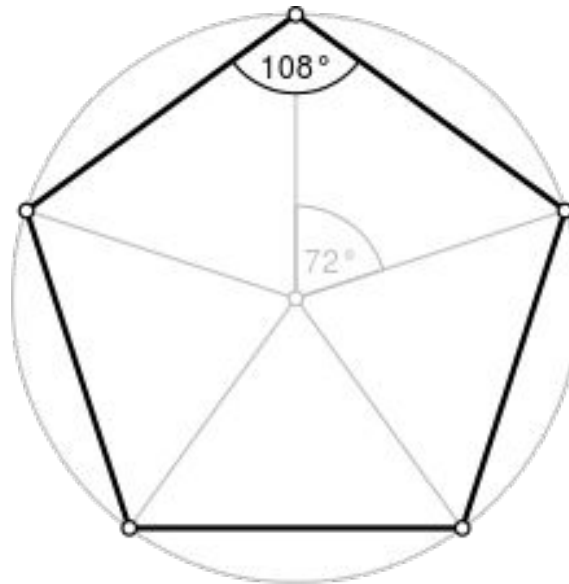
- $X = X + \text{Distância} * \cos(\text{Ângulo})$
- $Y = Y + \text{Distância} * \sin(\text{Ângulo})$

Conversão graus para radianos:

$$\text{GRAUS} * \pi/180.0 = \text{Rad}$$

Exercício

Desenhar um pentágono com cada segmento medindo 30 unidades de medida utilizando somente a função de desenho de linha.



Tela estática

Resolveremos o problema de tela estática utilizando a função `glutTimerFunc` (que tem a assinatura abaixo), ela cria um gatilho que dispara a função passada por parâmetro depois da quantidade de msecs passada por parâmetro.

```
void glutTimerFunc(unsigned int msecs, void (*func)(int value), value);
```

Criaremos então uma função chamada `redraw`:

```
void redraw(int value);
```

```
int delay = 10;
```

Então na função `main` chamaremos:

```
glutTimerFunc(delay, redraw, 0);
```

E então criaremos a função `redraw` do seguinte modo:

Tela estática

```
void redraw(int value) {  
    ◦ glutPostRedisplay();  
    ◦ glutTimerFunc(10, redraw, 0);  
}
```

Onde `glutPostRedisplay()` é a função que chama a função de desenho definido e depois colocamos mais um timer para a função “se chamar” depois de 10 milisegundos, fazendo um loop.

Alguns includes também

```
#include <math.h>
```

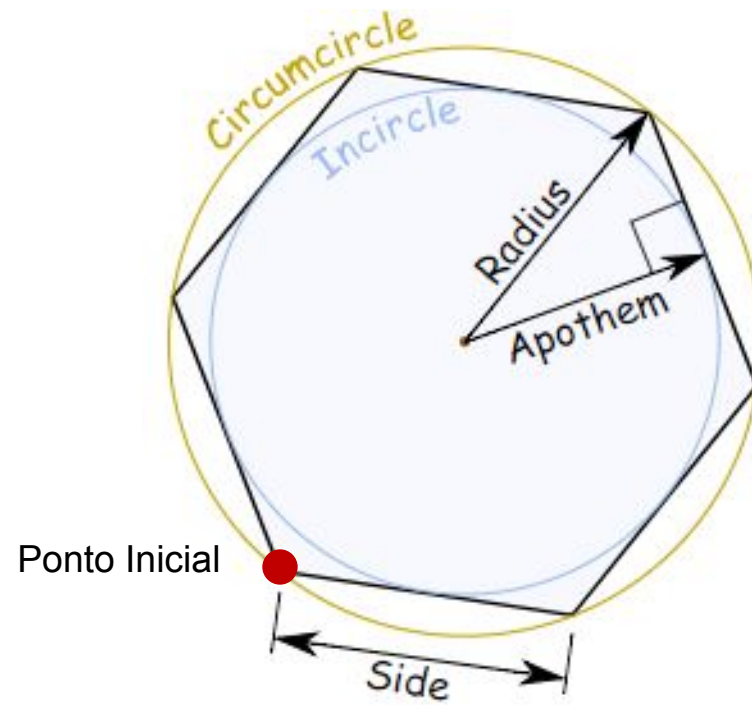
```
#include <vector>
```

```
using namespace std;
```


Polígono

```
class Poligono {  
    ◦ double tamanhoLado;  
    ◦ int numLados;  
    ◦ pair<double, double> posicao;  
    ◦ pair<double, double> escala;  
    ◦ double rotacao ;  
  
    ◦ Poligono (double tamanhoLado, int numLados, pair<double, double> posicao);  
  
    ◦ desenhar();  
    ◦ deslocar (double distancia, double direcao);  
    ◦ escalonar(double escalaX, double escalaY);  
    ◦ rotacionar (double direcao);  
};
```

Polígono



Calcular o Apothem

$$\text{apothem} = \frac{s}{2 \tan\left(\frac{180}{n}\right) \pi}$$

Funções

Poligono(double tamanhoLado, int numLados, pair<double, double> posicao){

- `this->numLados = numLados;`
- `this->tamanhoLado = tamanhoLado;`
- `this->posicao = posicao;`

- `this->escala = pair<double, double>(1, 1);`
- `this->rotacao = 0;`

}

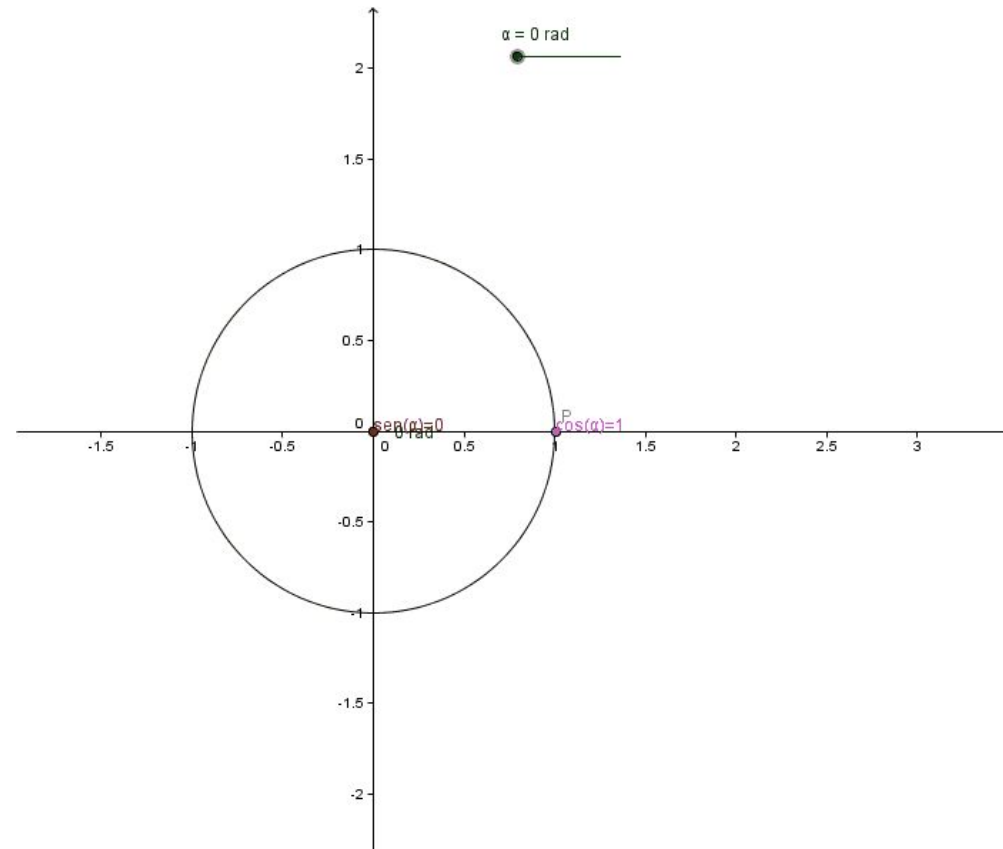
Funções - Desenhar

- **desenhar(){**
 - `pair<double, double> vertice;`
 - `vertice.first = posicao.first - tamanhoLado / 2;`
 - `vertice.second = posicao.second - tamanhoLado / (2 * tan(3.1416 / double(numLados)));`

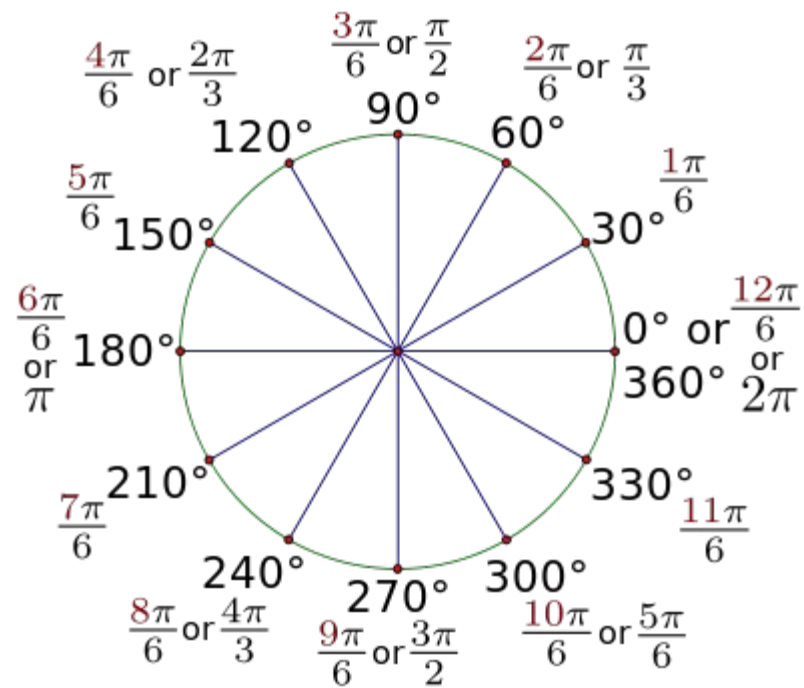
 - `//calcular o ângulo`
 - `//360 / numLados`
 - `//convertendo para radianos (2 * pi) / numLados`

 - `//deslocar os próximos polígonos e desenhar as linhas;`
 - `//X = X + (Deslocamento * Direção)`
 - `//cosseno para X, seno para Y`
- `}`

Direção



Direção



Funções - Deslocar

A função para deslocamento é simples, só precisamos alterar o ponto de origem pelo seguinte parâmetro:

- $X = X + \text{Distância} * \cos(\hat{\text{Ângulo}})$
- $Y = Y + \text{Distância} * \sin(\hat{\text{Ângulo}})$

Funções - Escalar

A função para escalonamento é simples, só precisamos alterar a escala, depois modificar isso na função de desenho.

Funções – Rotação

O passo mais simples é aumentar o valor da variável rotação e aplicar esta rotação inicial na função de desenho, para ao invés de iniciar a primeira reta com o ângulo 0, utilizar este ângulo.

Mas este processo somente faz o polígono girar ao redor do primeiro ponto desenhado e não ao redor do centro, para isto precisaríamos rotacionar o primeiro ponto. Três dicas para o desafio de fazer o polígono rotacionar em relação ao seu centro:

- Toda rotação deve ser feita a partir do ponto de origem 0,0. Podemos deslocar a origem até 0,0 e voltar ao ponto anterior depois de rotacionar.
- Podemos fazer a rotação pelo processo geométrico básico:
 - $X' = X * \cos \theta - Y * \sin \theta;$
 - $Y' = X * \sin \theta + Y * \cos \theta;$