Tanque de Provas Numérico da USP

Eric Henrique Favero

Rubens A. Amaro Junior

Manual Prático BURNER

Prof. Dr. Cheng Liang Yee

São Paulo

2017

**Pré-processador BURNER[[1]](#footnote-1)**

Sumário

[Burner 3](#_Toc498785840)

[main.cpp 5](#_Toc498785841)

[ioside.cpp 8](#_Toc498785842)

[Point.h 11](#_Toc498785843)

[ioflow.cpp 13](#_Toc498785844)

[Particle.h 15](#_Toc498785845)

[define.h 16](#_Toc498785846)

[Arquivo de entrada .txt 18](#_Toc498785847)

[Arquivo GRID 19](#_Toc498785848)

[Exemplos 20](#_Toc498785849)

[Visualização no ParaView 24](#_Toc498785850)

[Observações e Cuidados importantes 28](#_Toc498785851)

# Burner

Pré-processador gerador das posições espaciais (x, y, z) definidas para diferentes materiais (ID’s). Através de funções matemáticas no código computacional, representadas por comandos (*if, while, for,* ...), o programa gera o domínio computacional discreto (geometria do problema) utilizado pelo *solver* baseado no método de partículas *Moving Particle Semi-implicit* (MPS).

O **Burner** é um programa escrito em linguagem C++, composto por seis arquivos:

**Cabeçalho (*Header*)**

define.h

Particle.h

Point.h

**Fonte (*Source*)**

main.cpp

ioside.cpp

ioflow.cpp

Os arquivos podem ser editados e o programa compilado com diversos editores, como por exemplo:

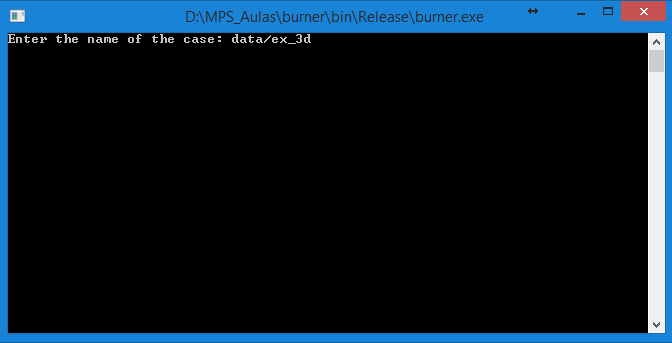
**CodeBlocks -** [**http://www.codeblocks.org/**](http://www.codeblocks.org/)

**NetBeans -** [**https://netbeans.org/**](https://netbeans.org/)

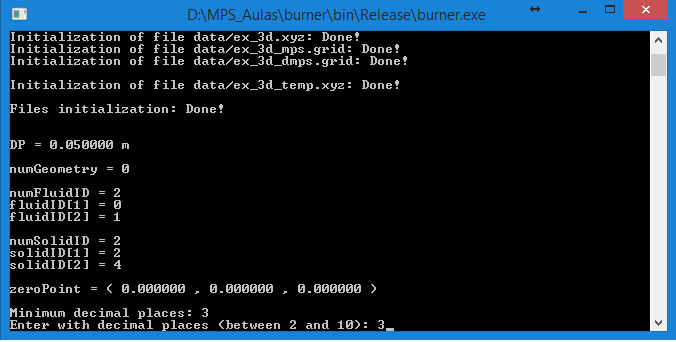
**Dev-C++ -** [**http://orwelldevcpp.blogspot.com.br/**](http://orwelldevcpp.blogspot.com.br/)

Na pasta do **Burner**, já existe um **project** pré compilado para o **CodeBlocks**, o arquivo *burner.cbp*.

Após compilação do código, através da sua execução é aberto um **prompt** de comando onde deve ser passada a pasta e **nome do arquivo (extensão txt) com dados de entrada** (geralmente data/nome do arquivo):



Após ENTER, o programa apresenta as entradas do arquivo de entrada e pergunta o **número de casas decimais** para impressão das posições geométricas do modelo. O usuário deve escolher a precisão que deseja e o programa começa a gerar o modelo, mostrando os arquivos de saída à medida que são finalizados.



Para visualização das geometrias geradas pelo **Burner** (arquivo .xyz), pode-se utilizar o *software*:

**ParaView -** [**http://www.paraview.org/**](http://www.paraview.org/)

Basicamente, o **USUÁRIO** terá de alterar, os arquivos **ioside.cpp** e **ioflow.cpp** para gerar as geometrias.

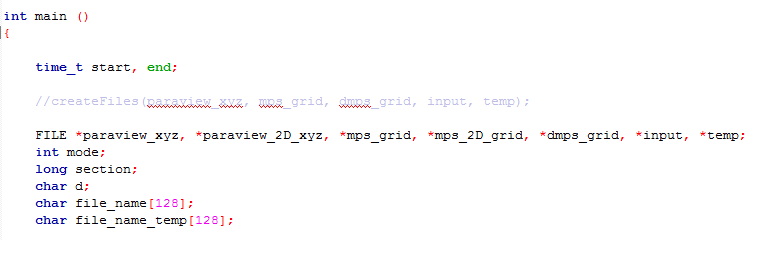
Dependendo da geometria, os arquivos **define.h** e **Particle.h** também poderão ser alterados.

A seguir são apresentadas as principais funcionalidades dos arquivos que compõem o **Burner**.

# main.cpp

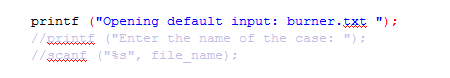
O arquivo **main.cpp** contem a estrutura principal do código, ou seja, os principais comandos realizados durante a execução. Abaixo são apresentadas algumas das principais partes do arquivo. Esse arquivo, praticamente, não deve ser alterado.

Nas primeiras linhas do arquivo, são declaradas algumas variáveis do *int main*.

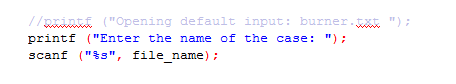


Um arquivo de entrada (extensão txt), é acessado pelo programa. Através do comentário de algumas linhas (//), pode-se escolher entre:

* Programa sempre lê um arquivo nomeado “burner.txt”

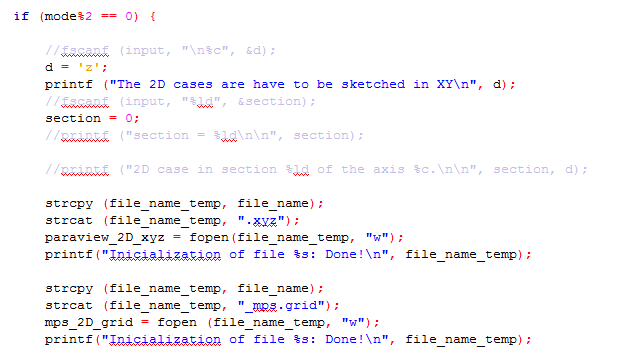


* Programa lê arquivo com nome dado pelo usuário



O **nome do arquivo de entrada** é usado em todos os **arquivos de saída** gerados no programa.

Após a passagem do nome do arquivo, o programa verifica se o modelo é bi ou tridimensional:



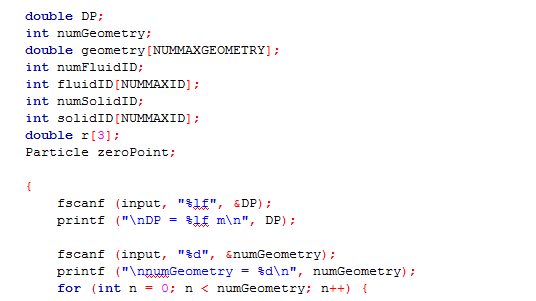
Criando arquivos nos seguintes formatos:

.xyz: arquivo com tipo de partícula (ID) e posições (x,y,z) para visualização no PARAVIEW.

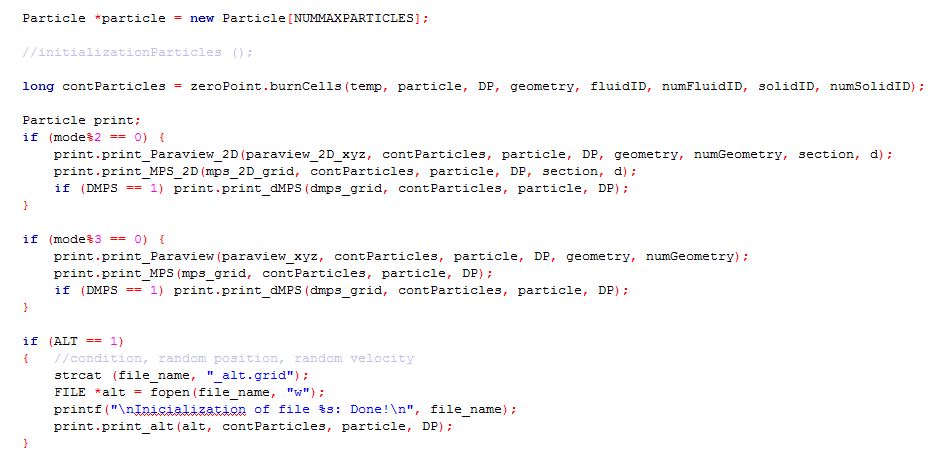
.grid: arquivo com tipo de partícula (ID), posições (x,y,z), vdelocidades iniciais (vx, vy, vz), pressão inicial e PND inicial. Arquivo passado como entrada do *solver* MPS.

Um arquivo com sufixo \_temp.xyz também é gerado. Caso ocorra algum problema ou o programa não tenha terminado sua execução, esse arquivo pode ser visualizado no PARAVIEW (http://www.paraview.org/), contendo a parte do modelo gerada até o momento.

Novas variáveis são definidas e recebem valores do arquivo de entrada:



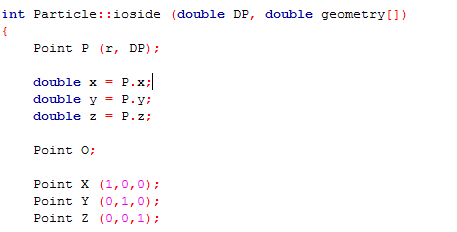
O pograma chama a estrutura *Particle*, que possui as principais funções para construção do modelo, define a variável *countPartcles* (número de partículas) e realiza a escrita dos arquivos de saída:



# ioside.cpp

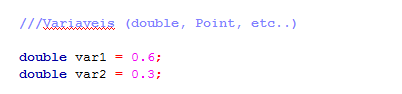
Arquivo onde o **USUÁRIO** define **dimensões** e **equações** que delimitam as geometrias.

Primeiro são definidos pontos e variáveis. (NÃO ALTERAR ESSA PARTE !!!)

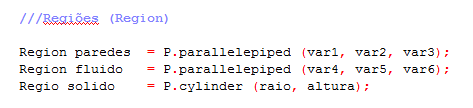


**DP** representa a distância entre pontos, ou **distância entre partículas**, lido pelo **Burner** do **arquivo de entrada**.

**Variáveis** que serão utilizadas para gerar as geometrias são **definidas** pelo **usuário**.

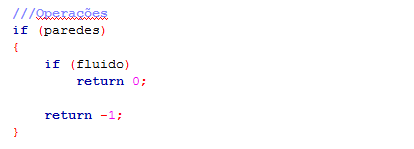


**Regiões** representando as principais geometrias são **definidas** pelo **usuário**.



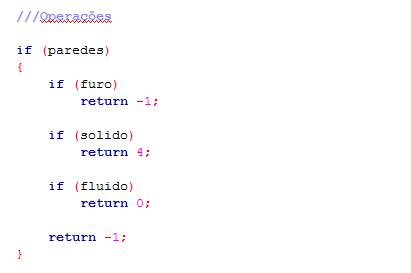
O arquivo **Point.h** contém algumas funções ***geometrias*** e ***transformações*** (translação e rotação) prontas para serem utilizadas. Outras geometrias e transformações podem ser criadas pelo usuário e colocadas no arquivo **Point.h** ou então, colocadas diretamente no arquivo ioside.cpp, com utilização de *if*, *else*, *for,* ....

**Operações** para construção das geometrias são **definidas** pelo **usuário**.

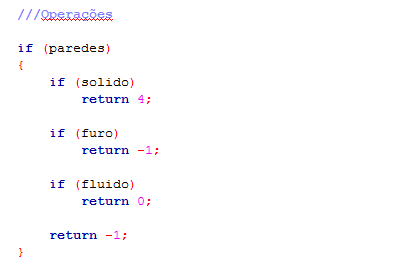


Cada ***return*** ***ID*** se refere a um tipo de material diferente (fluido ou sólido), conforme passado pelo arquivo de entrada. Para definirmos uma **região vazia**, colocamos ***return* -1**.

ATENÇÃO, TODAS AS GEOMETRIAS SÃO GERADAS SEGUINDO A PRIORIDADE EM QUE FOI ESCRITA NO ARQUIVO. Por exemplo, para gerar um sólido com furo, dentro de um fluido, as chamadas devem ser definidas na seguinte ordem:



Caso o usuário deixe as chamadas em ordem diferente, como por exemplo:



O sólido estará dentro do fluido, porém o furo no sólido não será feito, pois o programa já terá gerado todo o sólido, não permitindo a criação do furo (espaço vazio).

Os sólidos terão a parede externa com ID = número passado no arquivo de entrada e pelo *return*, e 2 camadas de paredes internas (*dummies*) com ID = número passado no arquivo de entrada e pelo *return* + 1.

Caso seja necessária a geração de um sólido com paredes internas e externas em contato com fluido, e portanto, mínimo de 4 camadas intermediárias de *dummies* nos casos 2D e 2 camadas intermediárias de *dummies* nos casos 3D, operações de união ou subtração podem ser utilizadas.

Uma limitação do **Burner** é a criação de sólidos para casos 2d. O BURNER NÃO CONSEGUE GERAR SÓLIDOS INTERNOS PARA CASOS 2D. O usuário deve realizar operações de subtração para gerar as paredes dummies e espaços vazios.

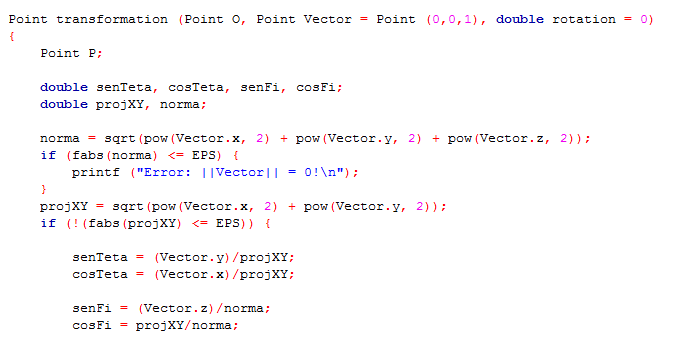
Após as operações, deixar ***return 2*** para construção de paredes que delimitam todo o domínio físico. Parede interna ID = 2 e parede externa ID = 3 (*dummies*).



# Point.h

Principais geometrias e transformações pré-definidas no programa. O usuário pode criar uma nova geometria e adicionar no arquivo.

Exemplos:

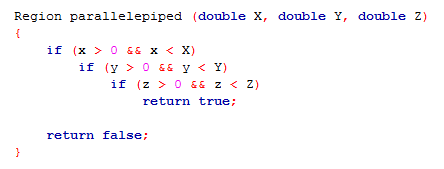


Operações de translação e rotação sobre uma geometria.

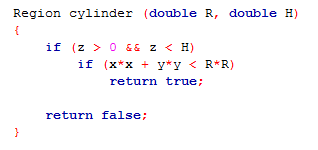
Point O – Translação.

Point Vector – Vetor onde será realizada a rotação.

double rotation – ângulo de rotação em graus.



Paralelepípedo com dimensões X, Y e Z.

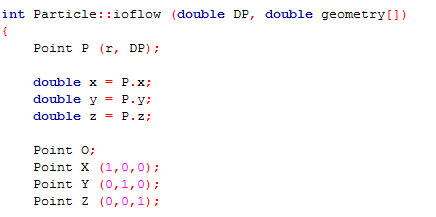


Cilindro de raio R e comprimento H.

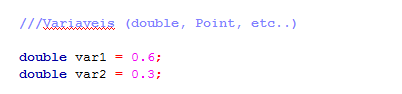
# ioflow.cpp

Eventuais cortes e transformações sobre as geometrias e/ou paredes externas.

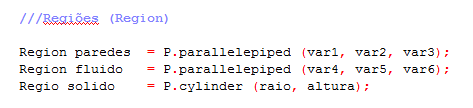
Primeiro são definidos pontos e variáveis. (NÃO ALTERAR ESSA PARTE !!!)



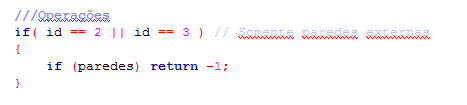
Variáveis que serão utilizadas para cortes nas geometrias são definidas pelo usuário.



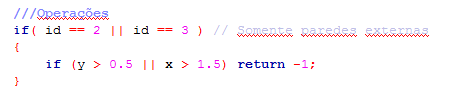
Os cortes podem ser definidos por regiões.



E colocados nas operações:

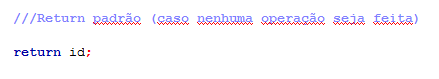


ou então, colocados diretamente nas operações com utilização de *if*, *else*, ...:



Além de paredes externas, como apresentadas nos exemplos acima, outros materiais podem ser removidos.

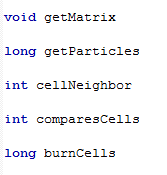
Após as operações, deixar ***return id***.



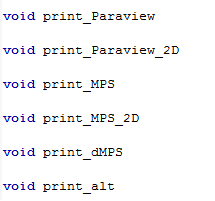
# Particle.h

Além das funções detalhadas anteriormente como ***ioside*** e ***ioflow***, dentro da estrutura *struct Particle* existem outras funções:

* Responsáveis pelo **sistema de busca** e **construção do modelo**.

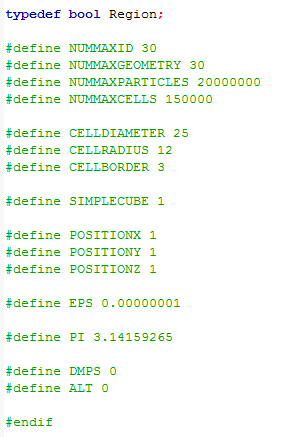


* Responsáveis pela **escrita** nos **arquivos de saída**.



# define.h

Parâmetros básicos como dp, ids, ponto zero.



Variável booleana Region.

NUMMAXID - Número máximo de materiais (fluidos ou sólidos diferentes).

NUMXAGEOMETRY - Número máximo de geometrias (Ainda não implementado).

NUMMAXPARTICLES - Número máximo de pontos que o programa pode gerar.

NUMMAXCELLS - Número máximo de células utilizado pelo programa para geração do modelo.

CELLDIAMETER –

CELLRADIUS –

CELLBORDER –

SIMPLECUBE –

POSITIONX -

POSITIONY -

POSITIONZ -

EPS - Precisão das variáveis.

PI – Valor de PI.

DMPS - Gera (1) ou não (0) arquivo de saída para o DMPS. Somente para casos 3d.

ALT - Gera (1) ou não (0) arquivo .grid para inclusão de condições no xml que serão utilizados no solver MPS. Útil para automatizar a inclusão de condições de um número grande de partículas.

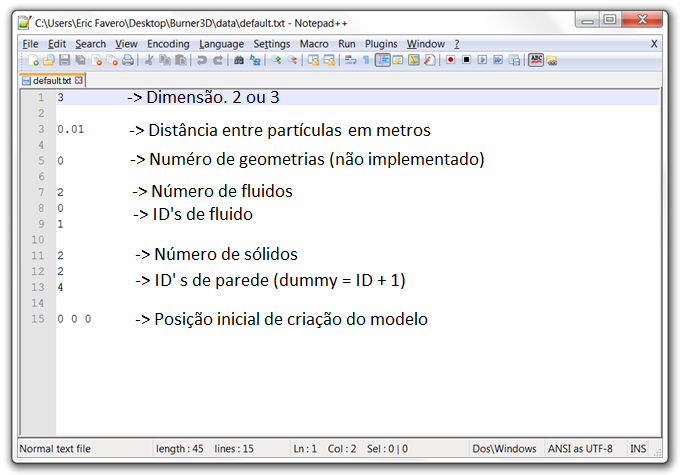
# Arquivo de entrada .txt

O arquivo de entrada ***nome\_do\_arquivo.txt***, possui os principais dados de entrada passados para o **Burner**.

O NOME DO ARQUIVO **NÃO** PODE CONTER ESPAÇO, como por exemplo “*meu arquivo.txt*”. Ao invés disso, podemos nomear o arquivo utilizando traço “meu\_arquivo.txt”.

A figura abaixo apresenta o que cada linha do arquivo de entrada irá passar para o **Burner**.

ATENÇÃO ESPECIAL PARA A POSIÇÃO INICIAL DE CRIAÇÃO DO MODELO!!! O USUÁRIO DEVE PASSAR A COORDENADA DE UM PONTO PRÓXIMO A UMA REGIÃO ONDE EXISTA O MODELO (FLUIDO OU SÓLIDO) QUE SERÁ GERADO. CASO NÃO EXISTA UM PONTO DO MODELO PRÓXIMO DESSA POSIÇÃO, O BURNER NÃO GERA O MODELO.



# Arquivo GRID

Arquivo de texto gerado pelo **Burner** e que servirá como entrada do solver MPS.

1a linha: Número zero (default)

2a linha: Número de pontos (partículas)

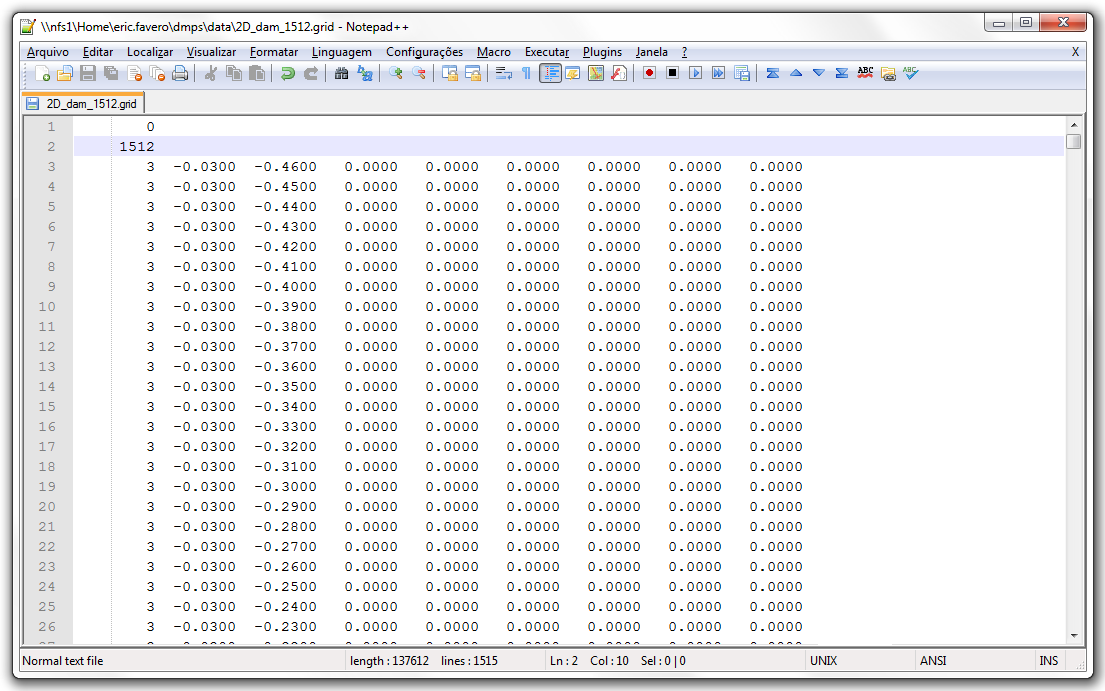
1a coluna: tipo de material (ID)

2a, 3a, 4a colunas: posições (x,y,z)

5a, 6a, 7a colunas: velocidades iniciais (vx,vy,vz)

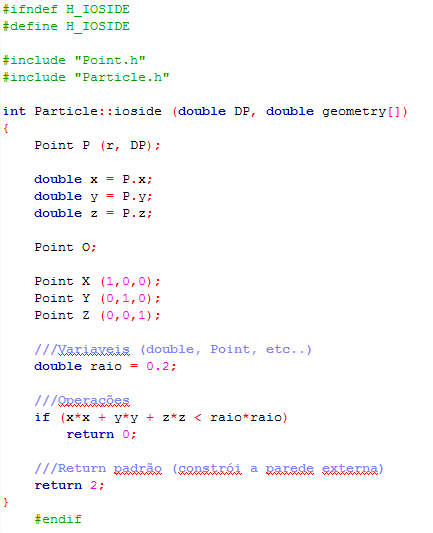
8a coluna: pressão inicial

9a coluna: PND inicial

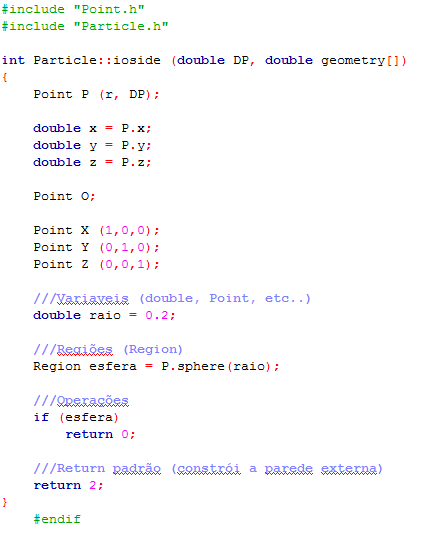


# Exemplos

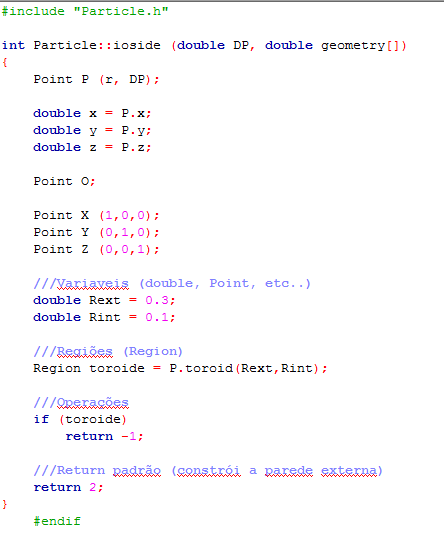
Neste exemplo será gerada uma esfera de raio 2 m (x² + y² + z² <= 2²) completamente preenchida com fluído de ID = 0.



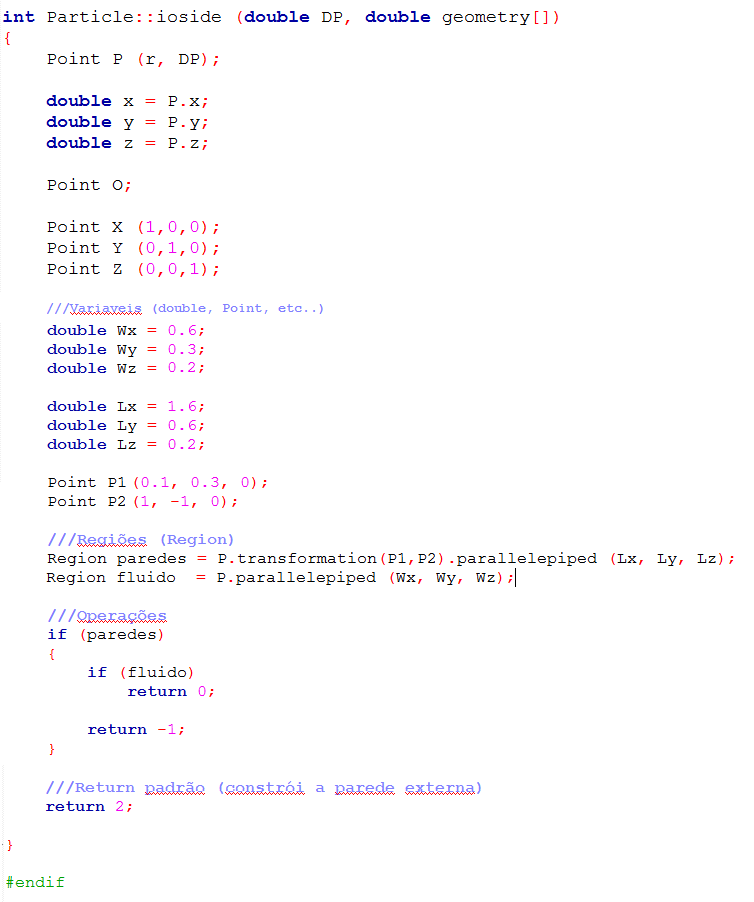
O mesmo exemplo poderia ser feito com uma função de geometria *sphere*



Neste exemplo será gerado um toróide vazio (ID = -1).



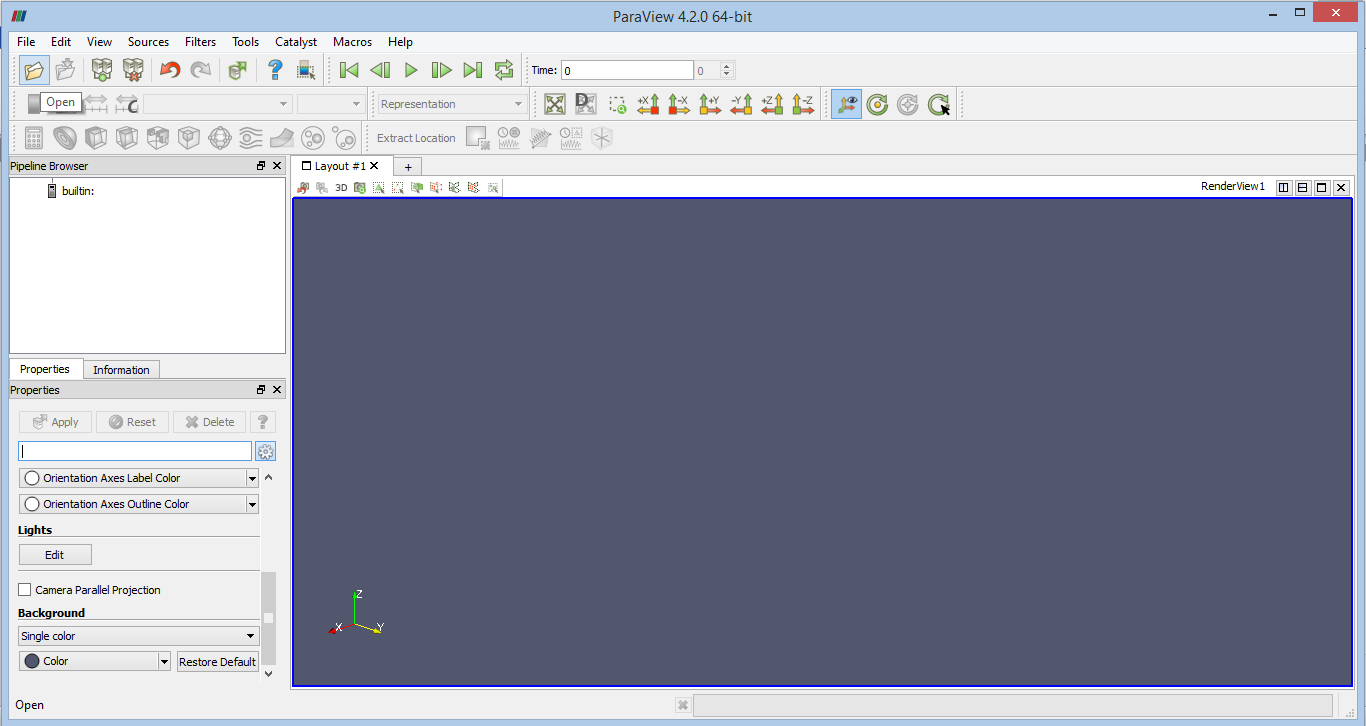
Neste exemplo será gerado uma caixa retangular deslocada da origem (0,0,0), uma distância P1 = (0.1, 0.3, 0) e rotacionada - 450, ou seja, base paralela ao vetor P2 (1, -1, 0). Ao invés do vetor P2, poderia ser informado o ângulo de - 450. A caixa será preenchida com uma coluna de fluido (ID = 0), de dimensões (0.6, 0.3, 0.2).



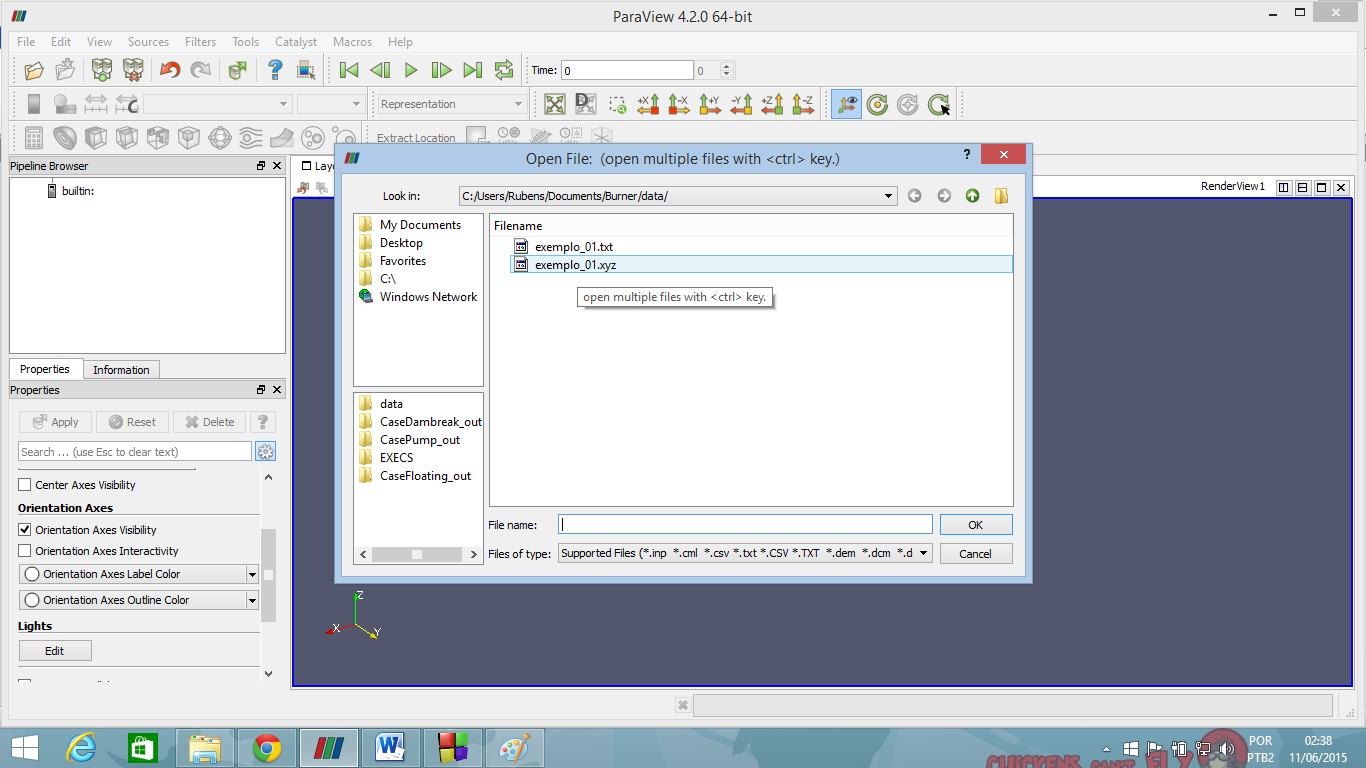
# Visualização no ParaView

Para visualizar um modelo no Paraview (versão 4.2.0. Para versões atuais, consulte o tutorial do Paraview), siga os seguintes passos:

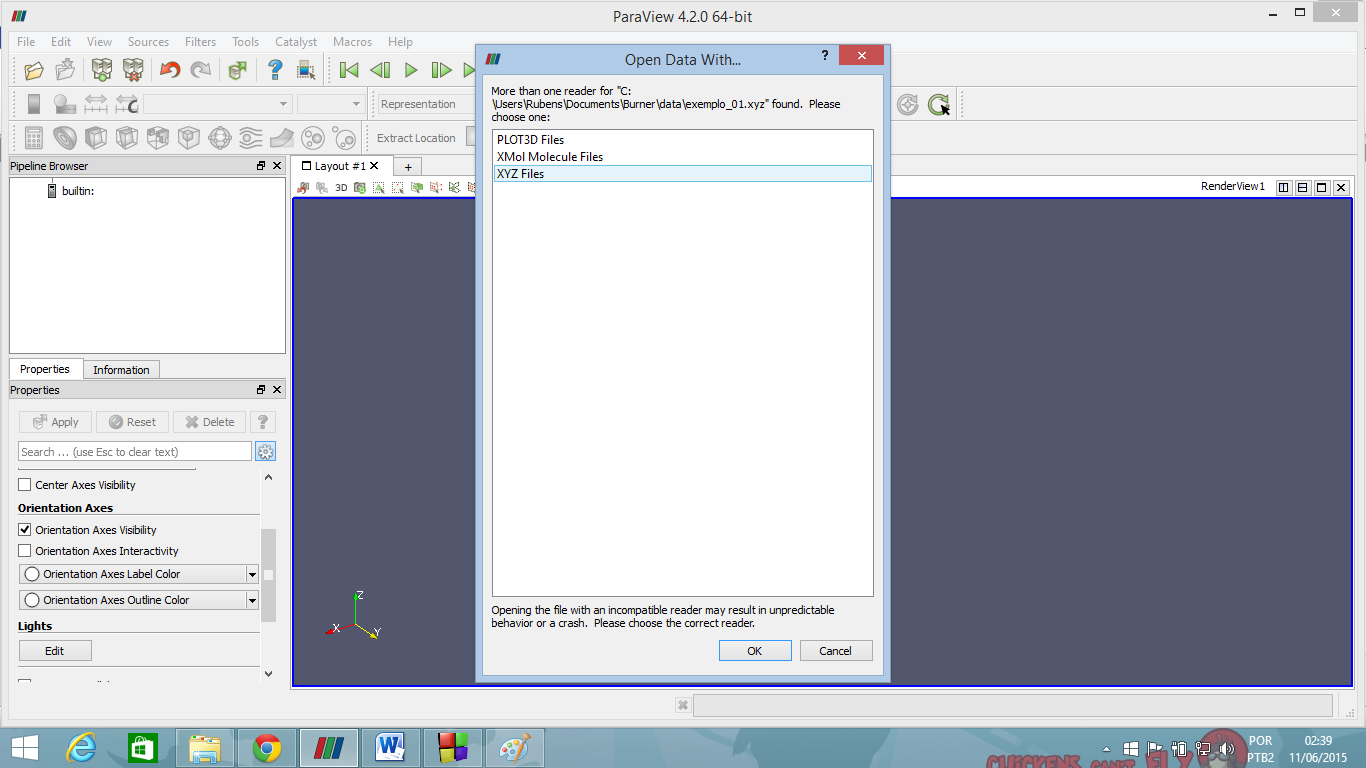
1. Selecione a pasta *Open* no canto superior esquerdo.



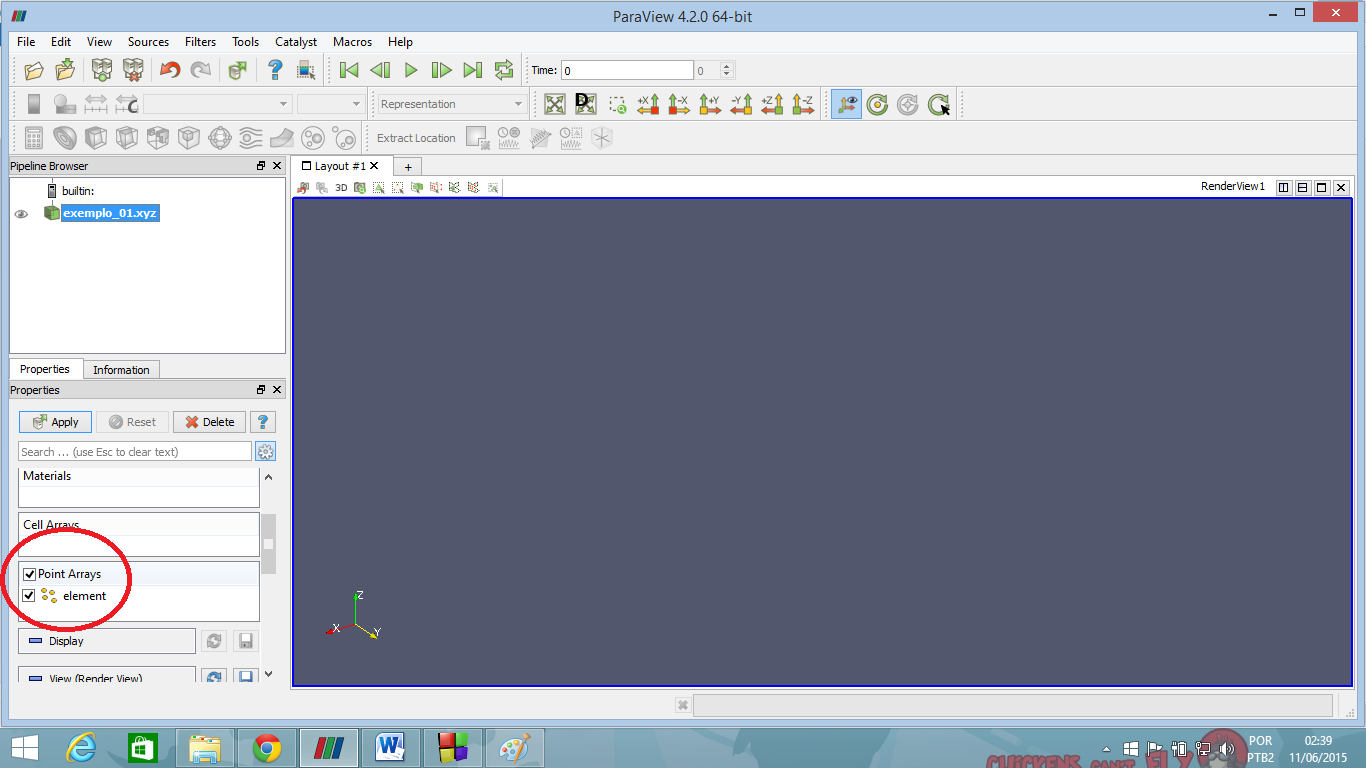
1. Selecione o arquivo .xyz.



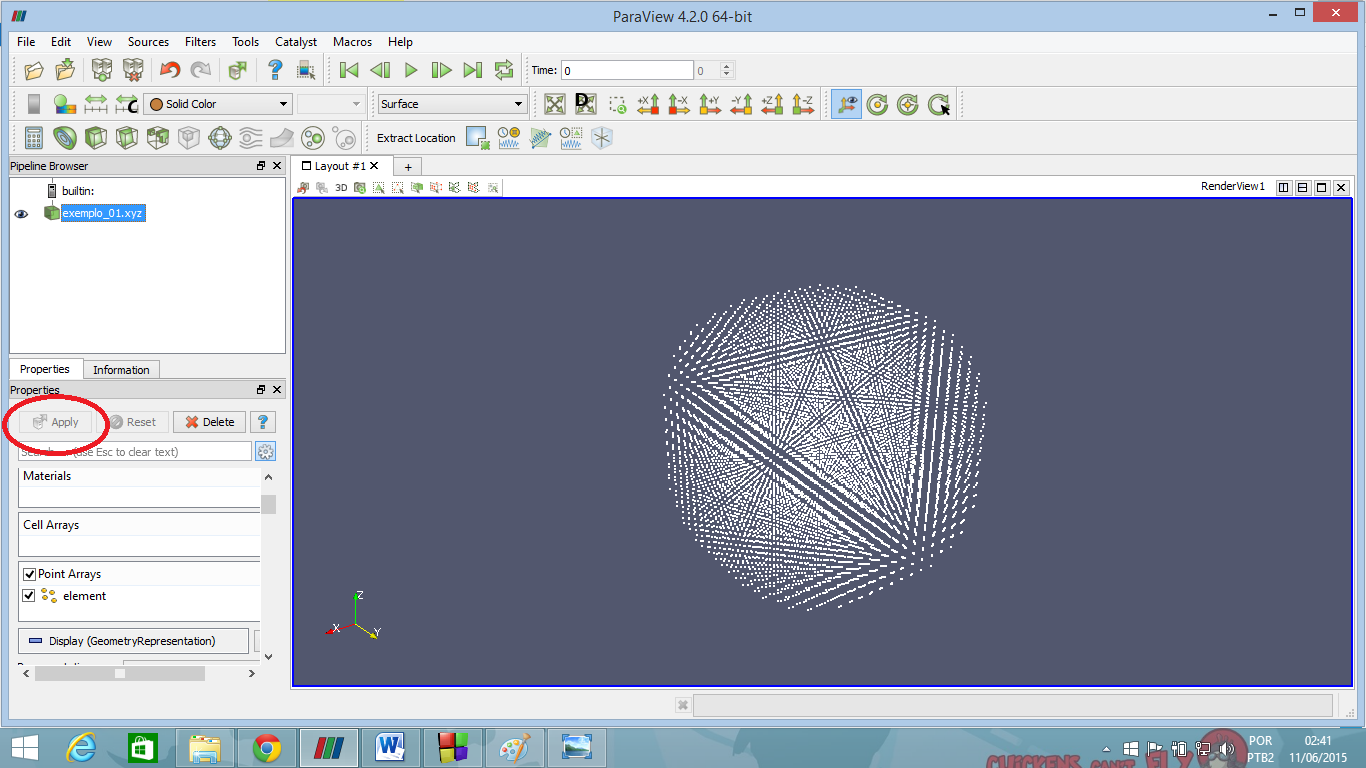
1. Selecione o formato XYZ *Files*.



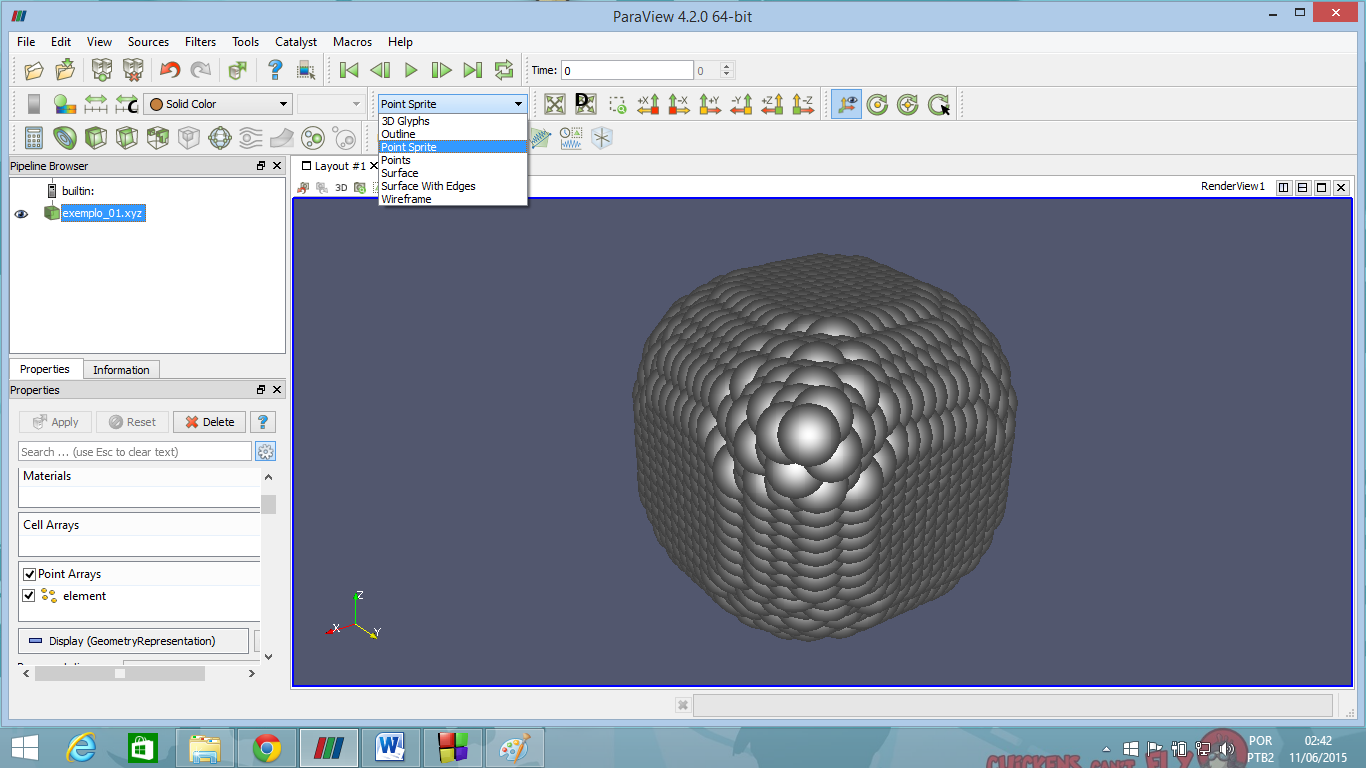
1. Habilite o campo *element*, para visualização de materiais em cores diferentes.



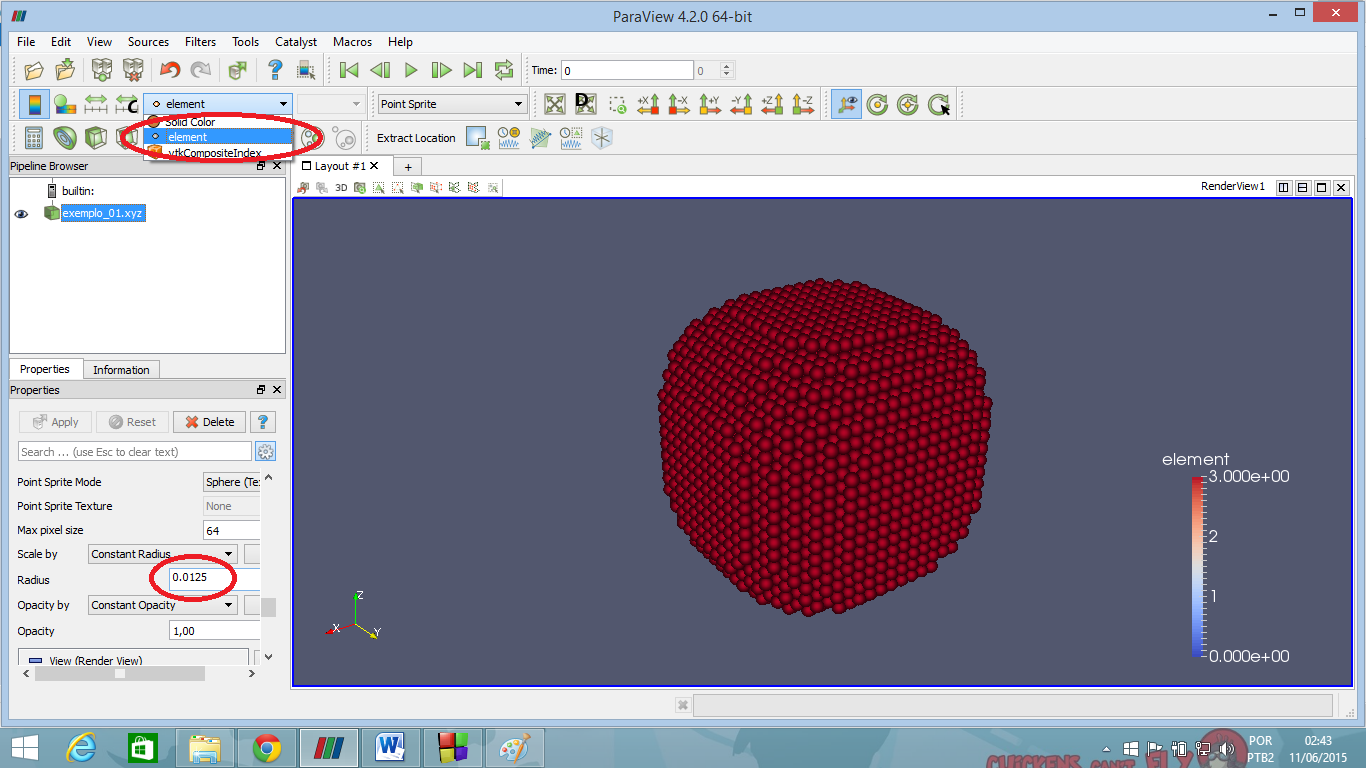
1. Click em *Apply.*



1. Para uma visualização melhor, podemos selecionar o modo *Point Sprite.*



1. Arrumar o raio das esferas, no campo *radius*, em função da distância entre partículas do modelo e selecionar o campo *element*.



Caso o modo ***Point Sprite*** não esteja habilitado, faça os seguintes passos:

*Tools -> Manage Plugins ... -> PointSprite\_Plugin*

Clique no botão ***Load Selected*** e marque o campo ***Auto Load.***

Reinicie o ParaView, abra um modelo .xyz e verifique se a opção Point Sprite está habilitada.

# Observações e Cuidados importantes

**Posição inicial no arquivo txt**

A POSIÇÃO INDICADA NA ÚLTIMA LINHA DO ARQUIVO DE ENTRADA TXT deve indicar um PONTO PRÓXIMO de uma região com FLUIDO ou SÓLIDO do modelo que será gerado. Caso não exista um ponto que represente fluido ou solido próximo dessa posição, o BURNER NÃO GERA O MODELO.

**Sólido interno nos modelos 2d**

O BURNER NÃO CONSEGUE GERAR SÓLIDOS INTERNOS PARA CASOS 2D. O usuário deve realizar operações de subtração para gerar as paredes dummies e espaços vazios.

**Precisão das casas decimais**

O **Burner** irá gerar as geometrias com um número de casas decimais em função da metade da distância entre partículas, ou seja, caso seja passado uma distância entre partículas de 0.005 m, o programa irá trabalhar com distâncias da ordem de 0.0025 m, 4 casas decimais.

O **Burner** está programado para trabalhar com valores entre 2 e 10 casas decimais. Se necessário, o usuário pode alterar esses valores nas funções de escrita dos arquivos de saída, localizadas no arquivo **Particle.h**. A precisão indicada nessas funções deve estar de acordo com a precisão que o **Burner** estará trabalhando.

Por exemplo, para uma distância entre partículas de 0.00005 m, (precisão de 0.000025 m, **6 casas decimais**) as funções devem estar no seguinte formato:



Ou seja, o valor após o ponto define a precisão dos números que serão escritos nos arquivos de saída. No exemplo, **%9.6lf**, indica seis casas decimais. A medida que utilizamos modelos com maior resolução, menores distâncias entre partículas, precisamos aumentar esses valores no **Burner**.

OBS: Não convém deixar um valor maior do que o necessário, pois isso acaba gerando arquivos com tamanhos cada vez maiores.

1. Software desenvolvido por Eric Henrique Favero.

   [eric.favero@gmail.com](mailto:eric.favero@gmail.com)

   Versão 1.0 [↑](#footnote-ref-1)