Blocked: An STL to block structure converter for faster prototyping and increased interchangeability of 3D printed parts

24-783 Spring 2023 Component Design

Unoptimized - Abraham George, Amirpouya Hemmasian, Charlotte Avra, Connor Geshan, Yayati Jadhav

Outline

No.	Component	Brief Description	Owner
01	**Voxelization: STL to Voxels**	Input is STL from user Output is voxel representation with 1x1 blocks only	Yayati
02	**Merge Function: Voxel to Custom Blocks**	Inputs 1x1 blocks and user preferences for block sizes Outputs representation with diff. block sizes	Abraham
03	**Stability Check**	Inputs “merged” representation Outputs true/false if any block is not supported	Charlotte
04	**Cost Functions**	Optimize for manufacturing cost? Lowest number of blocks possible? etc.	Pouya
05	**Graphical User Interface & OpenGL rendering**	Main app that the user interacts with, and renders the initial stl then renders blocked after merge. Fully functional	Connor
06	**Unit Tests**	Three tests per component	All

Overview

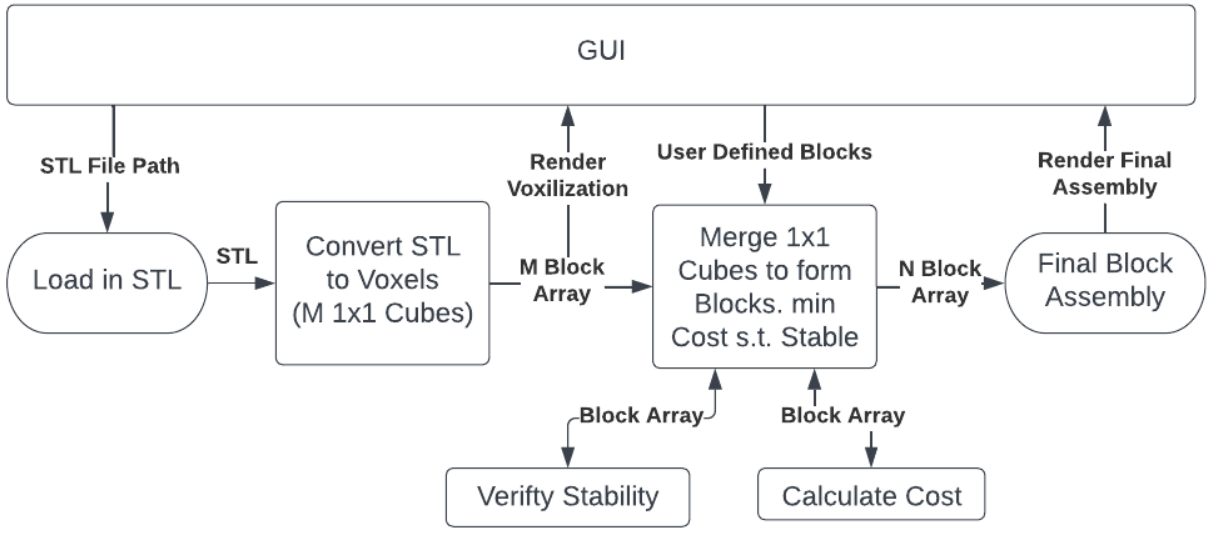

Figure 1: *Flowchart of Blocked*

01 Voxelization: STL to Voxels

- Input: STL file
- Output: (Block) N size array/matrix/vector
    - N = number of voxels
    - Blocks(float x, float y, int z(layer), int ID, bool isStable)
        - x, y, z -> center of 1x1 block
        - ID -> type of block, all 1 in this case. Using this data structure for consistency throughout the application
        - isStable: true or false if it is supported

Voxelize.h

```
void ReadInSTL( wxString str)
void GeneratePC()
std::vector<Block> CreateVoxels()
```

---

02 Merge Function: Voxel to Custom Blocks

- Takes an M length vector of 1x1 Block objects and combines these 1x1 blocks to form blocks from the list of user-defined shapes.
- inputs: (block) M size array/matrix/vector, (array) user selected block options: default 1x1
- output: (block) N size array of Blocks
    - N = number of blocks
    - Blocks(float x, float y, int z (layer), int ID, bool isStable)
        - The blocks should be generated such that they minimize the cost of the structure while maintaining stability.
            - Uses the Verify Stability and Calculate Cost functions
    - If any part of the structure cannot be built stability (ie. unsupported blocks), those 1x1 blocks will be ignored and the rest of the structure (that can be built stability) will be constructed. In the final N length Block vector, the unsupported blocks will be represented as special type of Block, a 1x1 “unbuildable” block.

Merge.h

```
std::vector<Block> mergeBlocks(std::vector<Block>, &std::vector<PossibleShapes>)
```

---

03 Stability Check

The stability of the layer determines whether or not the structure can be built as a free-standing structure. Voxelizing the STL and converting each voxel into custom blocks shows how the model could theoretically be built, but the stability check is to ensure the blocked STL is valid for real-world scenarios. The stability we check in this component is mainly looking for unsupported overhangs: areas where a block is not supported on any

There are a few cases in which the blocked model would be determined to be unstable:

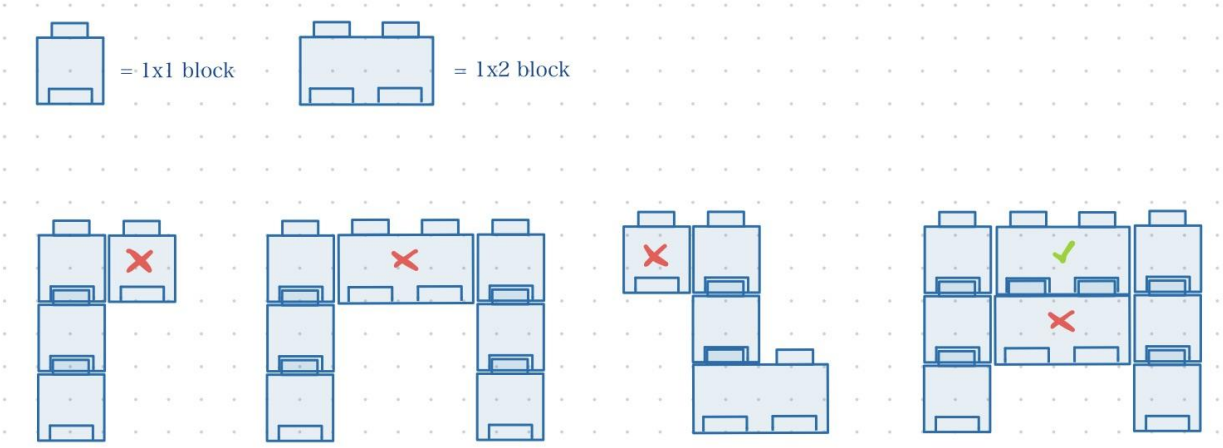

Figure 2: *Unstable block example cases (1-3 are possible, 4 is* ***not possible****)*

As shown in the figure, if a block is unstable, its ID will be updated to unstable and the layer will be marked as unstable. It is important to note that the stability function assumes all layers below the current layer are stable. Therefore, the case shown in the far right of the figure would never occur because the merge function would stabilize the second layer before moving on to the third layer. The merge and stability functions will act together to optimize each layer.

The main function of the stability check will:

- **Input**: all blocks of current layer, all blocks of previous layer
    - Assume: all previous layers are stable, otherwise the program would not have reached this layer
    - Check for any unsupported overhangs of current layer based on previous layer only
    - Update unstable block ID to unstable
    - **Return**: True if any blocks were found to be unstable
    - **Return**: False if no blocks anywhere in the current layer were found to be unstable

---

Stability.h

bool isLayerStable(current layer, previous layer)
bool checkLowerBlock(&current block, previous layer) // checks block directly underneath current

---

04 Cost Functions

- input: (blocks) M size array, user input cost function (manufacturing cost, number of blocks, etc)
    - optimize block choice based on given cost function
- return: (double) cost per layer
    - There can be different costs, such as printing cost, assembly cost, etc. The problem can be modeled as multi-objective optimization, or turned into a single-objective optimization, where the cost functions are added with certain coefficients that are the hyperparameters the user can pass to the function.

Cost.h

double cost(std::vector<Block> BlockList, &CostFunctionParameters, double cost_coeffs)

---

05 Graphical User Interface

- Implemented using wxWidgets
- this GUI has three internal components
    - App -> in charge of initializing the GUI and running the application
    - MainWindow -> in charge of all buttons, text fields, file input, bill of materials, and calling computational function listed below in the document
    - openGL canvas -> in charge of rendering the stl, and the block version of the stl post conversion
        - Render function (inside GUI code mentioned above):
            - input: (block) M size array
                - render each block and alter color based of block ID and stability

---

App.h

```
#include "wx/wx.h"

class App : public wxApp
{
public:
   bool OnInit();
};
```

MainWindow.h

```
#include "wx/wxprec.h"
#include "wx/wx.h"
#include "wx/gbsizer.h"
#include "wx/glcanvas.h"
#include "GLCanvas.h"

class MainWindow : public wxFrame
{
private:
   // Create a main panel, sizers, and OpenGL canvas
   wxPanel *mainPanel = new wxPanel(this);
   wxBoxSizer *mainSizer;
   wxGridBagSizer *gbSizer;
   GLCanvas *canvas;

   // Create a menu bar
   wxMenu *fileMenu = new wxMenu;
   wxMenuBar *menuBar = new wxMenuBar;

   // Saving helpers
   wxString CurrentPath;

   // Labels
   wxStaticText *stlInTextLabel;

   // Buttons
   wxButton *buildButton;
```

```
    // User Text Inputs
    wxTextCtrl *stlInFile;

    // User sliders
    wxSlider *vizSlider;

    // Block size options
    wxCheckBox *oneByOne;

    // Cost function options
    wxCheckBox *cFun1;

public:
    const int WIN_WIDTH = 800, WIN_HEIGHT = 600;

    MainWindow(wxWindow *parent, const wxString &title);
    virtual ~MainWindow();

    void CreateWindowLayout();
    void CreateWindowControls();
    void AddControlsToSizers();
    void BindEventHandlers();

    void OnNew(wxCommandEvent &evt);
    void OnOpen(wxCommandEvent &evt);
    void OnSave(wxCommandEvent &evt);
    void OnExit(wxCommandEvent &evt);

    void OnCloseClicked(wxCommandEvent &evt);
    void OnSaveClicked(wxCommandEvent &evt);
    void OnVoxelizeClicked(wxCommandEvent &evt);
    void OnResetClicked(wxCommandEvent &evt);
    void OnExportClicked(wxCommandEvent &evt);
    void OnChooseFileClicked(wxCommandEvent &evt);

};
```

GLCanvas.h

```
class GLCanvas : public wxGLCanvas
{
private:
    wxGLContext *glContext;

public:
    GLCanvas(wxPanel *parent, wxWindowID id = wxID_ANY, const wxPoint &pos =
wxDefaultPosition, const wxSize &size = wxDefaultSize, long style = 0, const wxString &name =
"GLCanvas");
    virtual ~GLCanvas();
```

```
    // Event handlers
    void resized(wxSizeEvent& evt);

        int getWidth();
        int getHeight();

        void render(wxPaintEvent& evt);
        void prepare3DViewport(int topleft_x, int topleft_y, int bottomrigth_x, int bottomrigth_y);
        void prepare2DViewport(int topleft_x, int topleft_y, int bottomrigth_x, int bottomrigth_y);

        // events
        void mouseMoved(wxMouseEvent& event);
        void mouseDown(wxMouseEvent& event);
        void mouseWheelMoved(wxMouseEvent& event);
        void mouseReleased(wxMouseEvent& event);
        void rightClick(wxMouseEvent& event);
        void mouseLeftWindow(wxMouseEvent& event);
        void keyPressed(wxKeyEvent& event);
        void keyReleased(wxKeyEvent& event);
};
```

---

06 Unit Tests

No.	Component	Description	Pass Condition
01	**Voxelization Test 1**	Check for empty file or other non stl format	Pass if provided file is an non empty stl file is provided
01	**Voxelization Test 2**	Check for stl type	Pass if provided file is binary stl and not ASCII stl or other stl file
01	**Voxelization Test 3**	Check if the provided binary stl has just one watertight mesh	Pass if the provided mesh is watertight and contains one connected mesh
02	**Merge Test 1**	Given a (fairly simple) voxelized shape that has been verified (by hand) to have a stable solution, generate the merged block list.	Will pass if the function returns a block list that is stable, and has a cost less than or equal to that we were able to achieve by hand for the test STL.
02	**Merge Test 2**	Given an empty block vector, return an empty block vector. Verifies that an empty vector doesn't cause an edge case crash	Function returns an empty block vector.
02	**Merge Test 3**	Given a (fairly simple) voxelized shape that does not have a stable solution (some part of the structure are not supported), return a block list	Will pass if it returns a proper block list for the parts that are able to be built stability, along with "unbuildable" 1x1 blocks for all of the regions that cannot be built.
03	**Stability Test 1**	If block is labeled unstable, check block directly underneath it	If there is no block underneath → pass
03	**Stability Test 2**	If block is labeled unstable, check isLayerStable returns False	If isLayerStable returns False → pass
03	**Stability Test 3**	If block is labeled stable, check isLayerStable returns True	If isLayerStable returns True → pass
04	**Cost Functions Test 1**	Given a predefined unit test set of blocks, calculate the cost.	Pass if the calculated cost equals predeifned const unit cost.
04	**Cost Functions Test 2**	Given an empy block vector, return the appropriate cost. Verifies it can handle the edge case of no blocks	Pass if it returns the cost of no blocks (likely 0).
04	**Cost Functions Test 3**	Set the cost parameters (user defined) to 0 and run the cost function on a specified geometry	Pass if it returns an appropriate cost value and doesn't encounter any errors (mainly divide by zero errors)
05	**GUI Test 1**	Can application be built on Linux, Mac OS X, and Windows	Pass if no build errors, user can open GUI on any platform

05	**GUI Test 2**	Successful rendering of STL file, if input file is correct.	Pass if STL file is render function is called without error and render appears on GLCanvas
05	**GUI Test 3**	Save functionality test, see if the user can export BOM and save new blocked data file.	Pass if non-empty, and correct files are saved to disk.