Tact-Di

Tactile Diagrams for the visually impaired

A Report Submitted
for the Project Assistant position

at

E-Health Research Center
International Institute of Information Technology
Bangalore

by

Medha Srivastava

under the supervision of
Professor T. K. Srikanth
in collaboration with
Vision Empower

July, 2018

Preface

Lack of accessible resources is a huge impediment for the education of the visually impaired. According to the 2011 Census of India, there are more than 11 lakh visually impaired children in the age group 5-19, 68% of whom attend school, 11.5% have dropped out of school and 20.5% have never attended school. The challenges in education for visually impaired students in India include their large numbers, dispersed distribution, diversity of languages in the country, and education systems across vernacular mediums. There are significant barriers especially in pursuing Science and Mathematics based courses, mainly due to unavailability of simple and effective ways to present mathematical notations comprising of equations, diagrams, and other complex interpretations to the student. This project tries to solve one such problem: visualization and understanding of Mathematics and Science diagrams for the visually impaired through the creation of Tactile Diagrams from images.

Contents

Chapter 1

Introduction

Diagrams or pictures provide an effective way of human communication. They are often used for representation as well as reasoning of certain types and hence play a vital role in subjects like Science and Mathematics. In the absence of vision, access of diagrams becomes a serious challenge for persons with blindness especially in subjects like Science and Mathematics where majority of concepts are conveyed through diagrams. It often leads to exclusion of a student in an inclusive classrooms where sighted students easily learn concepts through diagrams and a student with blindness is left with no option expect to image the diagram by listening or reading the related text. As a result, majority of students have to drop Science and Mathematics subjects in higher classes.

This challenge can be addressed by making tactile diagrams available to the visually impaired. Tactile diagrams are the raised representations of a visual image which is adopted for the sense of touch as shown in Figure 1.1. One or two tactile diagrams can save thousands of words. Over the decades, special educators have played a vital role of creating tactile diagrams. Diagrams are often created using art and craft materials and are extremely helpful in classroom setting for explaining various concepts to students with blindness. Creation of handmade tactile diagrams requires significant effort and time. Moreover, this method is not scalable especially when diagrams need to be created for thousands of books from different state boards across the country.

Figure 1.1: An example of tactile diagram

1.1 Objective

The objective of this project is to automatically convert simple images in Mathematics and Science to 3D or 2.5D models that can be 3D printed to create tactile models.

1.1.1 Why Automation

In an attempt to address the shortage of skilled designers, researchers have tried to automate the process of information translation from visual to tactile-suitable formats. Though there have been significant efforts to improvise the design and production processes for creating better quality tactile diagrams, the effort to automate these processes is in a very nascent stage. The desire for such automation can be understood in the context of countries like India, where there is a need to convert thousands of existing textbooks into a couple of dozen languages if the goals of universal education is to be attempted.

1.2 Related Work

Various efforts have been made towards creation of tactile diagrams:

- **Commercially available design softwares:** Like TactileView, TGD-Pro etc aimed at the creation of tactile graphics but limitations arise due to their generic nature with them being tied to a specific output method. Further drawbacks include their proprietary nature and a high learning curve.

- **PIAF Tactile Image Maker:** Piaf produces high quality tactile graphics using heat sensitive capsule paper. Piafs controlled heat source causes any black lines, letters or shapes that are drawn, printed or copied onto the capsule paper to swell. Limitations include manual creation of images to have different textures for different parts of the image in case of region drawings.

- **Edutactile:** Developed by the Assistive Technologies Group at IIT Delhi, Edutactile is a crossplatform software which automates the process of creation of tactile diagrams. The Image Converter module of Edutactile requires input image to be in the form of .svg, which makes it a limitation since automated conversion of an image to svg is often not accurate. Also, edutactile makes use of different texture assignment to differentiate between regions, which might work for simple region drawings, but for science diagrams close to the real world, it will include a lot of textures making the tactile diagram complicated and difficult to comprehend.

Few notable works have come from the field of computer science that attempt to develop decision-making frameworks and software applications for converting a scanned visual diagram into versions that are suitable to be made into raised line drawings. Accessible software tools with built-in design guideline frameworks for making tactile diagrams have also been proposed. Such tools can help educators and students make tactile diagrams themselves with minimal training. However, a commercially successful automation tool is yet to be seen.

Chapter 2

Approach

The first step was to classify the inputs into different problem classes:

- Line drawings
- Region drawings requiring different heights for different regions
- Colored drawings requiring custom heights for components

2.1 Line Drawings: Uniform height for foreground pixels

For line-drawings, we simply need to raise the foreground pixels to separate them from the background. Before any conversion we first need to perform some basic image cleaning operations.

- *Image pre-processing:*

  1. Conversion of image from colored to grayscale, since we are not concerned with colors in case of line drawings
  2. Invert the image in case foreground is lighter and background is darker, since the algorithm is built assuming that foreground will be lighter.
  3. Noise removal on the image: "Closing" morphological operation is suitable to account for any irregularities in the image

4. Resize the image according to a scaling factor if required

The above steps will bring the image to a standardized form for the conversion algorithm to work. The following two approaches work equally good and accurately for the conversion of line drawings to tactile models:

2.1.1 Blender-based approach

This approach converts 2D image into 3D model using Blender modelling. Blender provides the bpy module to the Python interpreter. This module can be imported in a script and gives access to Blender data, classes, and functions. So any operation on objects that we perform in 3D viewspace can also be done using python script. Following are the steps:

1. Remove all objects from the scene
2. Import the image (image file needs to be in SVG format)
3. Join all added objects if there are multiple curves in the scene
4. Scale the curve using a pre-defined scaling factor, since the SVG image imported into Blender scene will be extremely small
5. Convert the 2D image to 3D model by either of the operations:
   (a) Solidify object modifier: that takes the surface of any mesh and adds depth to it.
   (b) Extrusion of object in Z axis, assuming the curve already has appropriate thickness
6. Export the mesh to stl file

Since all the above steps can be done programmatically, Blender can be effectively used to automatically convert any line drawing to 3D model. However, for region drawings it is not possible to come up with an accurate generalized approach since individual components from SVG image are imported as curves in Blender.

2.1.2 Voxel-based approach

The idea here is to generate a 3D voxel data from 2D image data, and then convert the voxel data to 3D mesh. The steps can be broken down as follows:

- **Find the threshold for foreground/background separation:** The image that we have after pre-processing is still in grayscale. We need to separate the foreground from background so that the background can be at a lower level in the tactile model and the foreground can be raised. This is achieved using thresholding. We first normalize the image so that all pixel values are between 0 to 1, and then apply OTSU's thresholding method which will give an appropriate threshold for the image to separate foreground from background.

- **Creation of voxel data from image:** The voxel data from image is created layer by layer, where each layer can be considered as a cross-section of the 3D model about the z-axis. A line drawing with uniform height desired for the foreground will have the following 6 layers:

    1. Background layers (#1 and #2): These layers will contain all ones, i.e. all pixels will be raised to a minimum height for creating the base of the model.
    2. Image layers (#3 and #4): These layers will contain the same values as the image, so that after the conversion to mesh is done, these layers will be raised for foreground pixels and not raised for background pixels.
    3. Buffer layers (#0 and #5): These are the additional layers that will contain all zeros to act as zero padding for the marching cubes algorithm to work correctly, as explained further. Also, the boundaries of image layers and background layers should also be zero for the same reason.

- **Voxel to Mesh conversion:** *Marching cubes algorithm* is used to extract surface mesh from 3D volume. It computes the object bounds from the voxel data by forming an imaginary cube using eight neighbourhood locations at a time, and then determining the polygon needed to represent the part of the

isosurface (or mesh) that passes through that cube. Following params can be used to tune the marching cubes algorithm to generate desired mesh:

1. Level: This is the contour value that is used to search for isosurfaces in the voxel data. In our case this will be the threshold for the image that we calculated earlier.
2. Spacing: This parameter can be used to control how the voxel data will correspond to real world measurements. Spacing is a tuple containing 3 values for each dimension, hence to increase the height of the raised foreground features in the model, we can increase the z-value of spacing
3. Gradient Direction: This controls if the mesh was generated from an isosurface with gradient descent toward objects of interest, or the opposite. In our case this value will be 'descent', since foreground values are considered to be greater (lighter shade) than the background values.

The marching cubes algorithm will return the vertices, faces, normals, and values of the generated mesh or isosurface. This mesh can be exported as stl in the next step.

- **Mesh to STL conversion:** Numpy-stl provides a *mesh* module that can be used to create Mesh objects from a list of vertices and faces. This mesh can be written to an stl file using the *save* function.

Since voxel-based approach works at pixel level, it provides more control over different regions in the image. Hence, going forward, for all the other classes of diagrams, voxel-based approach has been used for the conversion.

2.2 Region drawings: multiple heights for different foreground regions

This solution is developed for images that are in grayscale and height can be assigned according to the grayscale value, i.e. a lighter pixel will be assigned a lesser height as compared to a darker pixel.

This implementation is a modification of the voxel-based approach for line drawings. While in line drawings a constant height was assigned to every foreground pixel, here the height will be assigned according to the value of pixel. The logic behind this implementation is to selectively replicate the image pixels across multiple layers. So if there are 10 layers in the model, the darkest pixel will be replicated across all 10 layers, while the lightest pixel will be replicated to only one layer. The number of layers in the model has to be predefined. If there are 'n' layers, the model will contain:

1. Base layer (#1): This layer will contain all ones, i.e. all pixels will be raised to a minimum height for creating the base of the model.

2. Image layers (#2 to #n+1): These layers will contain either one or zero value for a pixel depending upon the grayscale value for that pixel in the image. As explained above, a darker pixel will be replicated across more layers in the 3D coordinates for the model as compared to lighter pixel. Example: The value of a lighter pixel across the layers starting from the base might be something like [1,1,1,0,0,0,0,0,0], and for a darker pixel it might be [1,1,1,1,1,1,1,1,1,0].

3. Buffer layers (#0 and #n+2): These are the additional layers that will contain all zeros to act as zero padding for the marching cubes algorithm to work correctly. Also, the boundaries of image layers and background layers should also be zero for the same reason.

2.3 Colored drawings: custom heights for different foreground regions

There might be some cases where the grayscale value or the equivalent color to grayscale conversion is not a true representation of the height that has to be assigned to each pixel. In such cases we would need a custom height to be assigned to each component in the image, based on the use case. One such solution has been developed as a part of this project. Given an image, the algorithm identifies the major colors that are a part of the image, and presents to the user top 'm' colors

(m predefined). The user can then enter the relative heights for each color in the range 0 to 10 (0 default, i.e. a color for which height has not been assigned will assume the height equal to base of the model) as shown in 2.1. Since the heights are custom, this is a semi-automated approach.

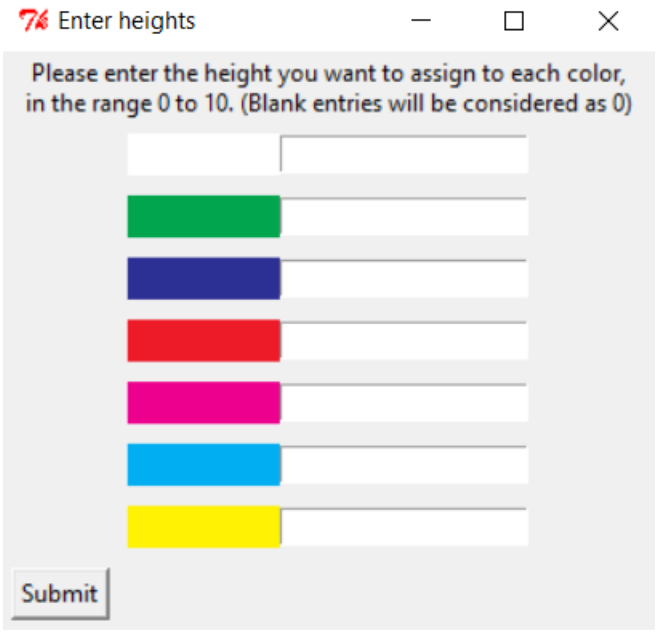


Figure 2.1: The interface where user can input heights for colors

The algorithm can be broken down as follows:

- **Identification of colors from image:** To pick up top prominent 'm' colors from image, we create a color frequency map for each pixel color in the image and then perform a grouping on the color frequency map to sum up frequencies for different color values which only vary slightly in their HSV values. Following are the steps:

    1. Iterate through all pixels and compute the frequency of each unique pixel color in a color frequency map
    2. Sort the color frequency map in the decreasing order of frequency, then pick up the top 100 colors

3. Perform deduplication for these top colors in HSV. To match a color against another, the deduplication algorithm considers a range for hue (-5 to +5), saturation(-10 to +10) and brightness (-10 to +10). Further it considers a threshold of 0.1% for color frequency and discards the colors having frequency less than that. The colors map will now contain the grouped colors with collective frequency.
4. Pick up the top 'm' colors from the above top colors map.

- **Height assignment to different colors:** For the top distinct colors extracted in the above step, the user is provided an interface where he can enter the relative height (ranging from 0 to 10) for each color.

- **Conversion of image from colored to grayscale:** After the distinct colors are mapped to height, we can iterate through the pixels in the colored image and convert the pixel value to gray value (ranging from 0 to 1) based on the height assigned. For example: if a pixel value is (238, 70, 58) and the height assigned to this color by the user is 8, then the corresponding gray value will be 0.1*8 = 0.8.

- **Grayscale image to model conversion:** After all the above operations, the problem boils down to Section 2.2, where we were converting grayscale image to 3D model. We can follow the same steps and get the model.

Chapter 3

Technical Specifications

3.1 Programming Language used

Python: Python is a widely used general-purpose, high-level programming language. Python is a very powerful language with a wide range of packages that can be easily imported into any project. Following python modules have been used in this project:

1. cv2: Opencv package to perform the image processing operations for noise removal, resizing, and morphological operations.
2. numpy: To deal with n-dimensional data and perform computations easily.
3. skimage: skimage contains the measure module which has the functions for marching cubes algorithm
4. stl: Contains utilities to convert mesh to stl
5. collections: For dictionary operations in python
6. Tkinter and tkFileDialog: For simple GUI that supports easy interaction and operations from the script
7. matplotlib: matplotlib.colors is used to convert from one color format to another
8. os: Input/output operations

3.2 Code components

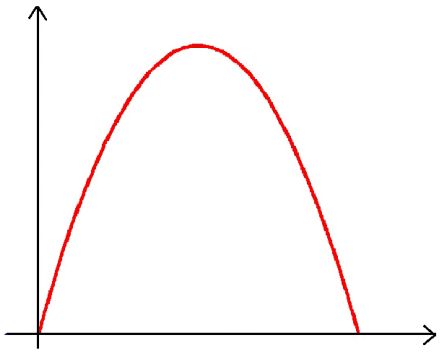
- **Init module:** Installs the required python packages. This is the first script and has to be run only once.
- **Main module:** This is the main module for image to model conversion. It calls the other utility modules to perform different operations.
- **Color extraction module:** Contains the utilities to get top distinct colors from the image. The algorithm for color matching and deduplication is a part of this module.
- **Color conversion module:** Contains the utilities to convert a pixel or complete image from hsv to bgr and vice versa in the desired format.
- **UI module:** Contains the utilities for displaying a simple GUI to the user where he can select image for conversion and input heights for different colors in the image.
- **File module:** Contains utilities for OS operations such as saving the mesh to a file.
- **Properties files:** Containing the file paths and image properties like height map and scaling factor.

Chapter 4

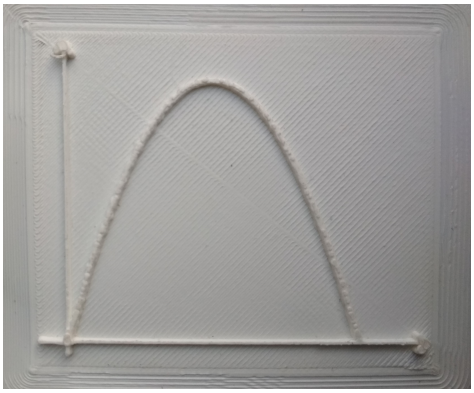
Result

The images were converted to 3D models and printed for different class of diagrams. Below are the images with their corresponding prints:

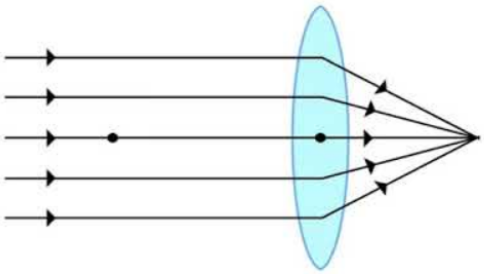
4.1 Line drawings:



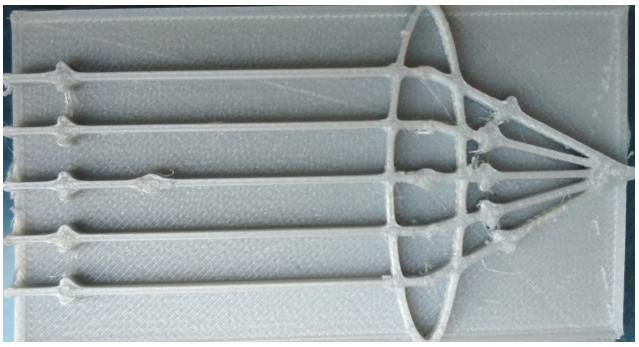
(a) Line drawing for parabola curve



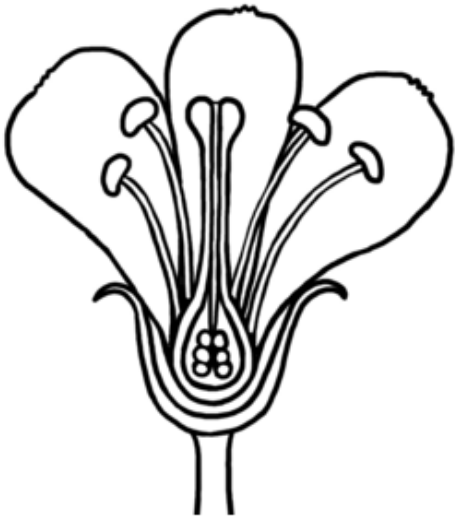
(b) Printed model



(a) Line drawing for lens convergence

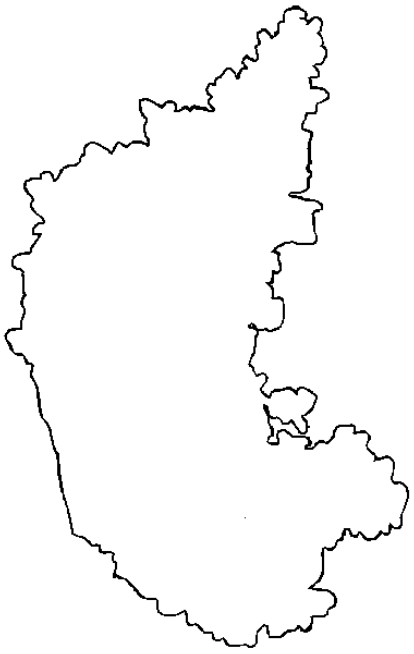


(b) Printed model



(a) Line drawing for parts of flower

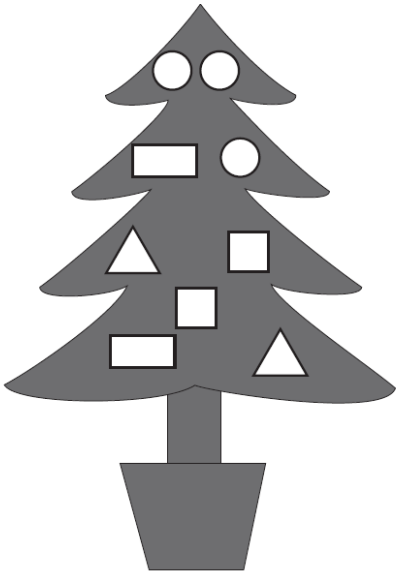
(b) Printed model



(a) Line drawing for map of Karnataka

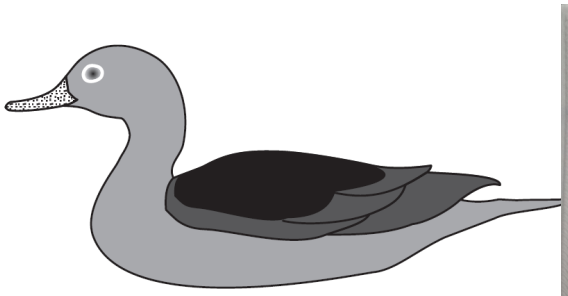
(b) Printed model

4.2 Region drawings for grayscale images:

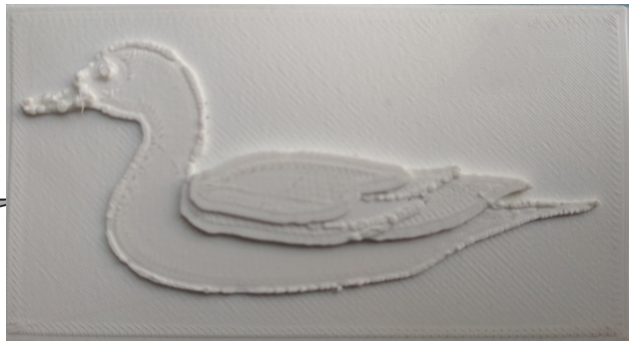


(a) Grayscale region drawing

(b) Printed model



(a) Grayscale region drawing



(b) Printed model

4.3 Region drawings for colored images:

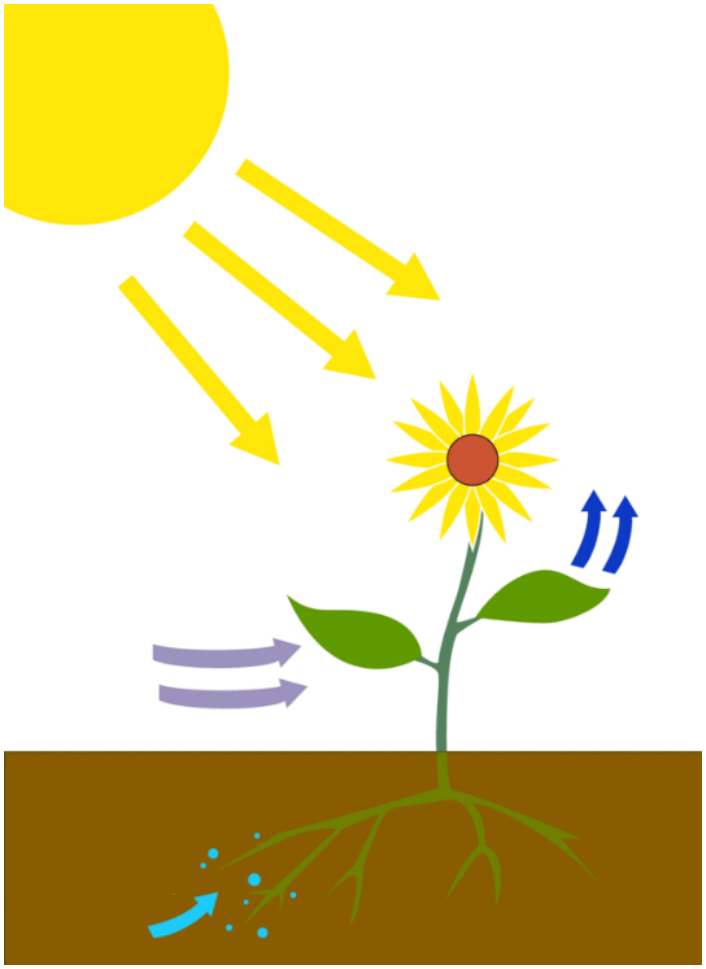


Figure 4.7: Colored drawing

Figure 4.8: Printed model

Chapter 5

Limitations

The current solution has been built and tested considering digital images as the input. Especially for colored image transformation, we assume that the colors are true, i.e. an object will contain only one color and there will be very less variation in the shade. This algorithm will fail to work in cases where an image component contains multiple colors/shades. Example: in the above diagram for photosynthesis, if sun is represented as a mixture of yellow, orange and red colors, this solution will not work.

For the same reason, the current solution might not work for textbook images which are hand painted drawings. It will also not work for labelled images since there is no way to separate text from the picture.

Chapter 6

Enhancements and Future work

1. For the colored image to model conversion, the interface where the user has to enter heights for different colors can be enhanced to show precomputed height values, which can then be changed by the user. This precomputation can be done by analyzing which part can be the foreground and which part can be a part of background (can be found by analyzing area occupied by a color).

2. The current algorithm assumes that different components of the image will have different colors, which might not be true for some images. This can be addressed by finding out the connected components of the image and assigning height to each component.

3. The height computation can further be automated by using machine learning techniques. The machine learning model can be trained on different class of images with their respective heights assigned in the current algorithm.

4. The height computation can also be automated if there is a way to identify the category of image and have heights preset for different parts of the image. Example: for mathematical graphs, if the axes has to be at a different height than the curve, an image can be recognized as a graph and components can be separated, then the preset values can be used to assign different height to the axes and curve.

5. Instead of assigning different heights to components, different textures can be assigned. This can turn out to be better for region drawings that are not close

to the real world images (and so the relative height between two parts does not matter).