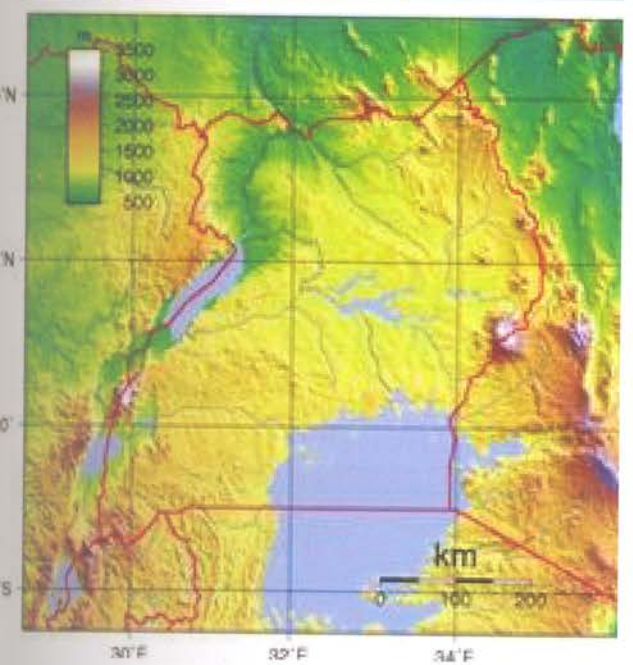
Topography Simulation

using depth scaling
over an approximate area of
1350 mm X 850 mm

Arnab Chatterjee
and
Shrikant P. Pathak

Background Information



A standard 2d topographical map

Topographic maps are based on topographical surveys performed at large scales. These surveys are called topographical in the old sense, showing a variety of elevations and landforms. This is in contrast to older cadastral surveys, which primarily show property and governmental boundaries. The first multi-sheet topographic map series of an entire country, was completed in 1789. The Great Trigonometric Survey of India, started by the East India Company in 1802, was notable for their successful effort on a larger scale and for accurately determining heights of Himalayan peaks from distant viewpoints. Topographic surveys were prepared first by the military to assist in planning for battle and for defensive emplacements as elevation information was of vital importance in such cases. However, as they evolved, topographic map series became a national resource in modern nations in planning infrastructure and resource exploitation.

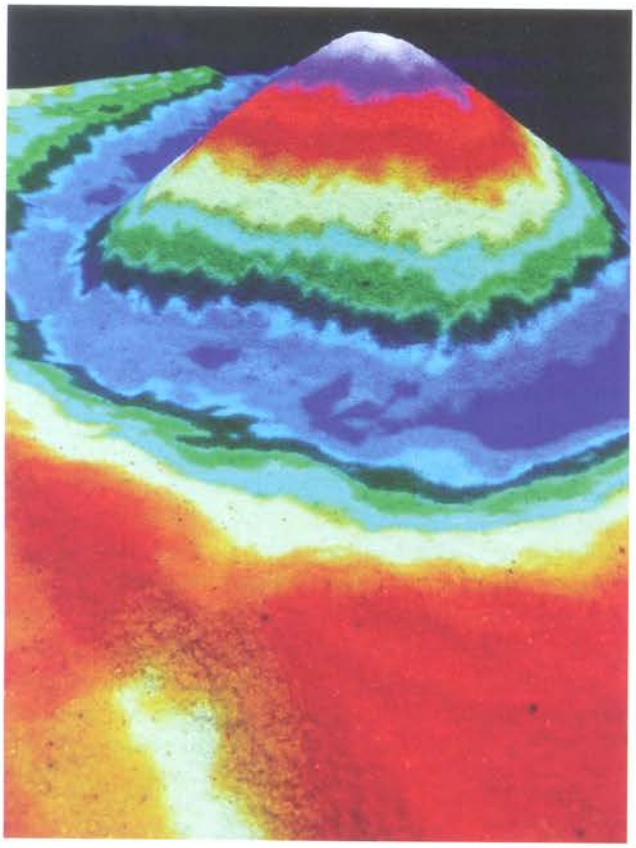
Topographic maps mostly being represented in two dimensional forms lack realistic representation of terrain or land relief. Three dimensional presentation of the topography on the other hand, helps in detailed understanding of land forms and geographic features. With the aid of remote sensing satellites, such mapping is achieved in reality. However, within the scope of non formal education in a Science centre, such attempts can help students to acquire deeper knowledge and better comprehension.

What it is –

A realistic ecosystem projected on top of a material surface emphasizes the variation in depth to recreate real-time model of a topological map which can reform according to the relief of the surface under consideration. However, the best part of the setup lies in its dynamic nature as it reforms the map as per changing topography. Hence it allows simulating any topography of need and thus analyzing the same as required.

Microsoft Kinect for windows is utilized here as a sensor to detect and scale a depth pattern in three dimension and thus to convey the data to a set of processing algorithm which determines the colour of a layer relieved at certain height, to project it back on the same surface with the help of a standard LCD projector. Starting from Physical modelling of topography to its analysis featuring an immediate reformation of the modified point cloud helps in understanding realistic behavior of a system targeted for simulation. In due course of the analysis we will plunge deeper into the working of the system and see how the challenges involved were overcome.

Propagation
A Journal of Science Communication



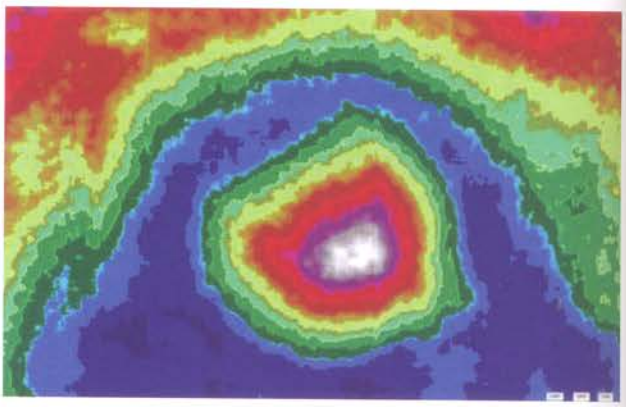
A realistic projection on a 3d terrain

The topography simulator can be described as a geographic map featuring changes in height, which can be manipulated and presented at a smaller scale yet with considerable degree of accuracy. As the map portrays detailed and accurate graphic presentation of natural and geographical features of the land it can even be generalized as a smaller scale chronographic map simulator where larger part of the ground is covered with reference to the elevation of the region.

What is new

a. Transformation from the regime of two dimensional maps to three dimensional mapped models

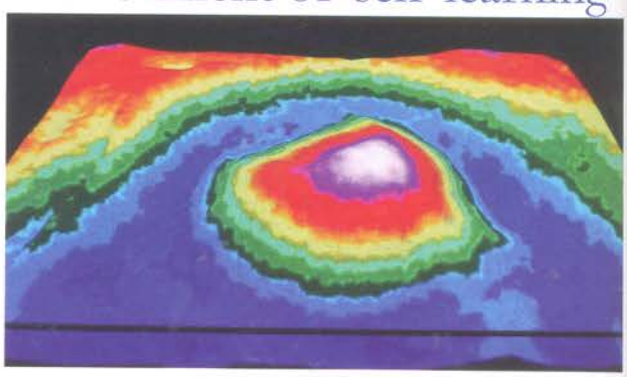
Two dimensional topographic maps in most of the cases fail to realistically portray the depth perception. A three dimensional presentation of the same help students or professionals understand the land forms



2D topography on a flat screen

assuring a higher level of comprehension. The maps, mostly being static in nature does not allow instant simulation of changes and thus to project implications so introduced. However the simulator can make a shift from the regime of static 2D maps to the era of dynamic 3D models while maintaining a great level of accuracy.

b. Utilization of Kinect in the field of education more specific to the science centre environment of self learning



Same topographic view projected on 3D material surface

Kinect, initially introduced to the consumers, was intended for gaming and solely entertainment. However, its remarkable combination of hardware, firmware and SDK allowed us to deploy the same for innovative projects. Keeping in view the dual nature of the system, within the arena of informal education in science centre and museums, it has evolved to be a gadget for futuristic presentation. Hence using Kinect under the realm of education is not as common as using the same in the field of gaming & entertainment.

c. Depth image processing in the context of surface profile determination

As the device was initially used for skeletal tracking and mostly being utilized for the purpose of determining human gestures, the impact of using depth frames solely had never been explored to this extent. However the capability of the device to remotely sense undulation of surfaces and capacity of .net framework to implement the same within the scope of windows presentation foundation is unique to some extent. Although the device solely depends on infrared spectrum which sometimes gets affected by random interferences even in indoor applications, a carefully selected band and minute filtration of data may help to prepare and reproduce a real life environment with much ease.

d. Simulation of unstable landscapes

One of the great features of this system is its dynamic behaviour and hence it allows simulation of unstable landscapes prone to land slide in hilly areas and soil erosion near sea shores. Hence it may help even professional planners and architects to simulate a landscape as per practical situation and understand the challenges before proceeding with final construction at the site.

How it works

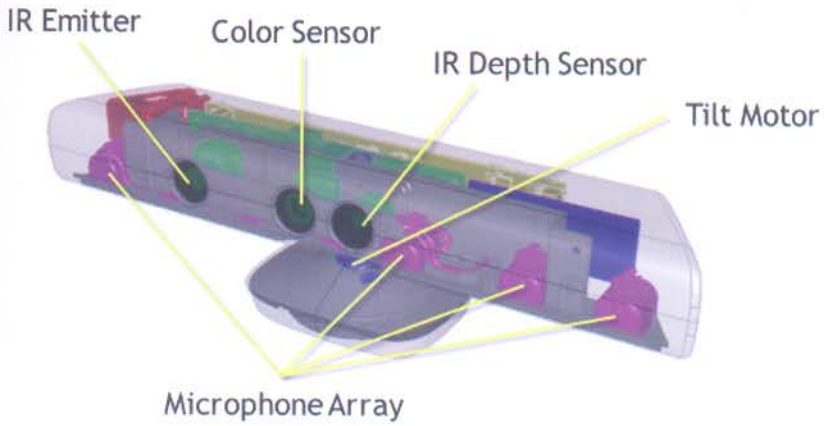
Building blocks of the system can be divided under three broad sub-heads which work hand in hand despite segregation as per their region of application.

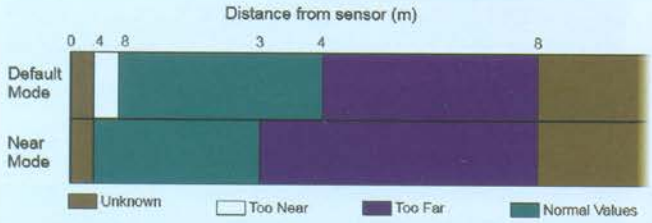
1. Microsoft Kinect for Windows

The innovative technology behind Kinect is a combination of hardware and software contained within the sensor itself that can be added to any existing computer. The Kinect sensor is a flat black box that sits on a small platform, usually horizontally; however in the application under consideration, it is mounted vertically. There's a trio of hardware innovations working together within the Kinect sensor:

- Color VGA video camera - This video camera aids in facial recognition and other detection features by detecting three color components: red, green and blue. Microsoft calls this an "RGB camera" referring to the color components it detects.
- Depth sensor - An infrared projector and a monochrome CMOS (complimentary metal-oxide semiconductor) sensor work together to "see" the room in 3-D regardless of the lighting conditions.
- Multi-array microphone - This is an array of four microphones that can isolate the voices of the players from the noise in the room. This allows the player to be a few feet away from the microphone and still use voice controls.

However within the scope of our discussion, we'll concentrate only on the depth sensor which generates depth data at a rate of 30 frames per second with a maximum resolution of 640 by 480 pixels. Unlike a web-camera, the depth sensor generates depth frames comprised of pixels which indicate distance between the sensor and the obstruction at a precision of approximately 3 millimeter within the range of 800 millimeter and 2000 millimeter. Still the hardware, though, would be erroneous without the breakthrough software that makes use of the data it gathers. The depth





data being received by the sensor is classified according to the following system –

Each pixel of the depth frame so received used to have 16 bits in it where 13 bits are used to represent the depth or distance of obstruction from sensor and left three for holding active player index. The details of the same may be presented as below –

Once depth data is received from the sensor over the high speed USB to the computer, depth bits from each pixel is filtered and further analyzed to map the terrain and undulations on the surface. Next the challenge lies in minimizing the error and non linearity present in the pixels so received. In the process of discussing the details of the application software we will understand how the same is minimized. Overall specification of depth data can be presented as below –

Property	Value
Angular Field of View	57° horz., 43° vert.
Framerate	approx. 30 Hz
Nominal spatial range	640 x 480 (VGA)
Nominal spatial resolution (at 2m distance)	3 mm
Nominal depth range	0.8 m - 3.5 m
Nominal depth resolution at 2n1 distance	1 cm
Deuce connection type	USB (+ external power)

2. An application software to analyze depth data received from the sensor –

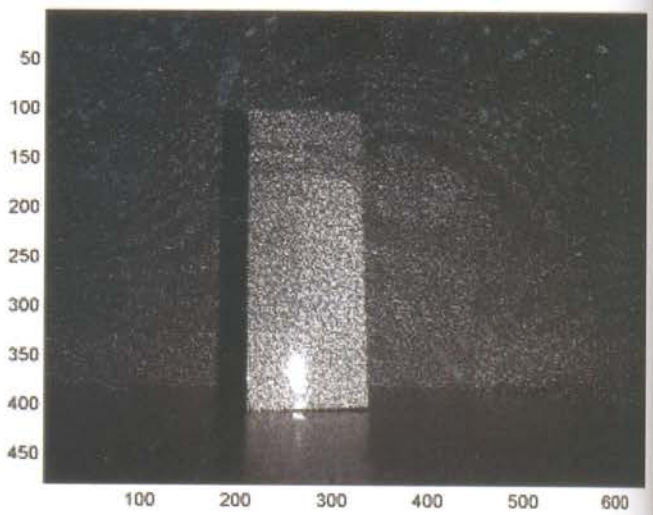
Once depth data is received in the form of depth frames which are 640 by 480 pixels generated at a rate of 30 frames per second by the Kinect, the first job is to filter the data. Broadly three types of filtration is adopted for realistic sensing. We will hence discuss the methods involved in such filtration routines one after the other.

Error and imperfection in the Kinect data may originate from three main sources:

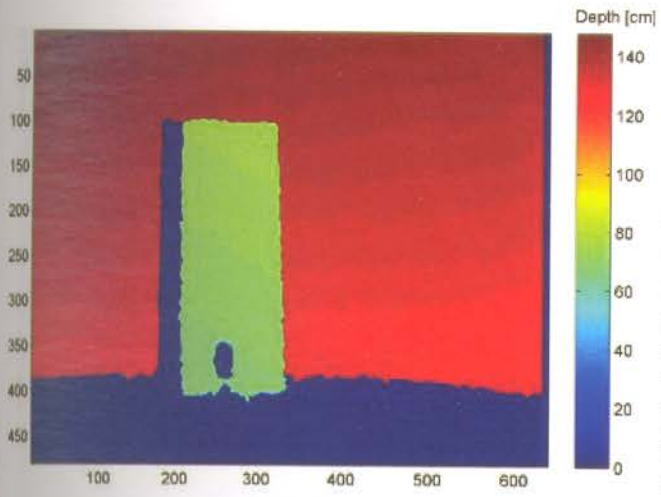
- The sensor;

- Measurement setup;

- Properties of the object surface.



(a)



(b)

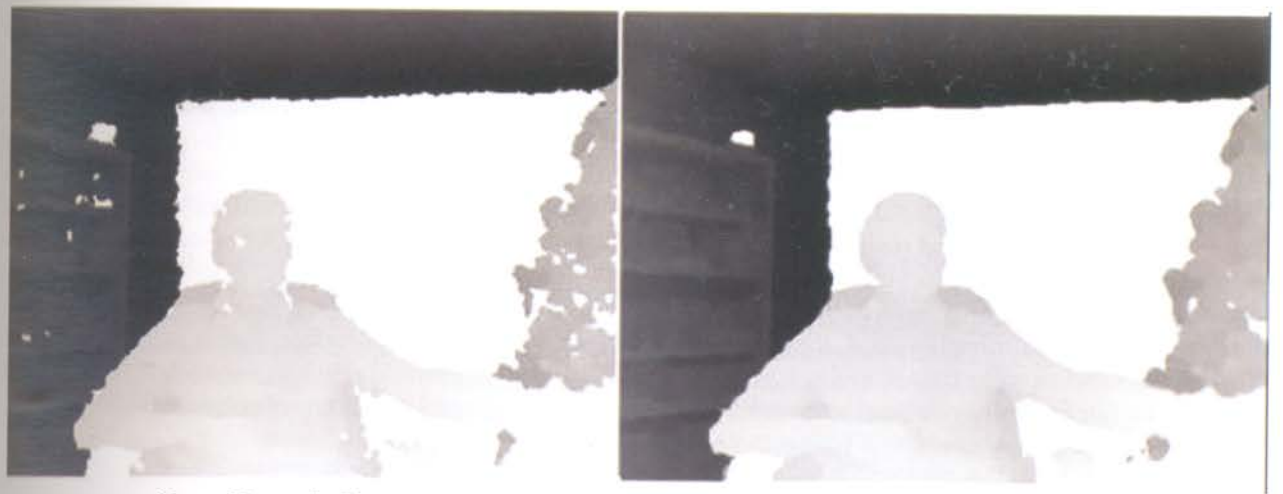
The sensor errors, for a properly functioning device, mainly refer to inadequate calibration and inaccurate measurement of disparities. Inadequate calibration and/or error in the estimation of the calibration parameters lead to systematic error in the object coordinates of individual points. Such systematic errors can be eliminated by a proper calibration as described in the previous section. Inaccurate measurement of disparities within the correlation algorithm and particularly the quantization of the disparities also influence the accuracy of individual points.

Errors caused by the measurement setup are mainly related to the lighting condition and the imaging geometry. The lighting condition influences the correlation and measurement of disparities. In strong light the laser speckles appear in low contrast in the infrared image, which can lead to outliers or gap in the resulting point cloud. The imaging geometry includes the distance to the object and the orientation of the object surface relative to the sensor. The operating range of the sensor is between 0.5 m to 5.0 m according to the specifications, and, as we will see in the following section, the random error of depth measurement increases with increasing distance to the sensor. Also, depending on the imaging geometry, parts of the scene may be occluded or shadowed. In Figure 1, the right side of the box is occluded as it cannot be seen by the infrared camera though it may have been illuminated by the laser pattern. The left side of the box is shadowed because it is not illuminated by the laser but is captured in the infrared image. Both the occluded areas and shadows appear as gaps in the point cloud.

The properties of the object surface also impact the measurement of points. As it can be seen in Figure 1, smooth and shiny surfaces that appear overexposed in the infrared image (the lower part of the box) impede the measurement of disparities, and result in a gap in the point cloud.

Removal of errors in depth measurement –

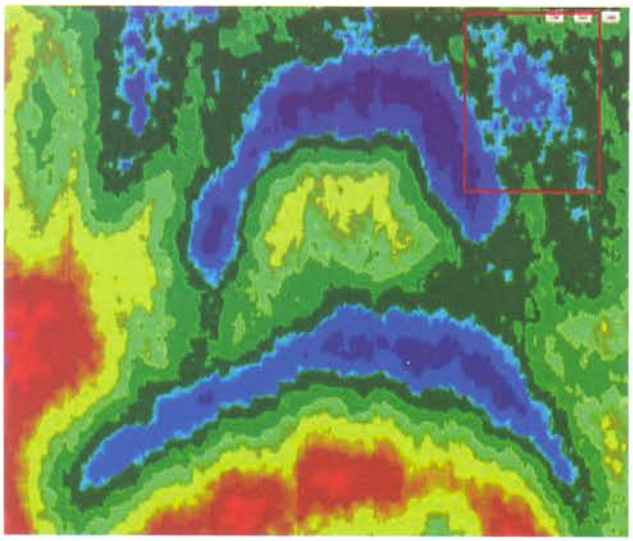
The first step in removing depth data discrepancy is making a space average of the pixels those cannot be identified by the sensor and hence marked as unknown. However precaution must be taken to differentiate between actual unknown values and unknown values adopted by the sensor by mistake. The process of segregation is calculated in a way that checks if an unknown pixel is surrounded by known pixel values of similar nature and if it happens a space averaged value of known pixels is placed at the position of the incorrect or unknown pixel. Shown below is the snap of a depth frame with minimized intermittent unknown pixels.



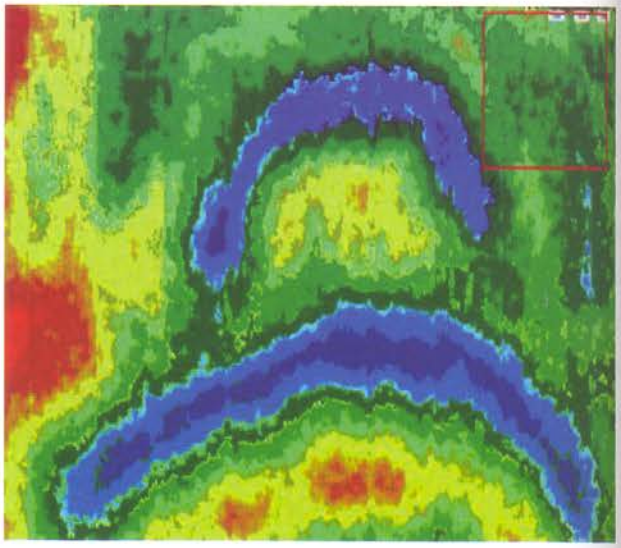
Raw Depth Image

Smooth Depth Image

Propagation
A Journal of Science Communication



Stray water bodies without filtration



Removal of stray water bodies after filtration

Instead of implementing complex algorithm to correct the depth data so received at the cost of valuable system resource, a dynamic adaptation technique is introduced. Before we started making complex reliefs on the surface, it is made as flat as possible. As kinect can measure depth right beneath the sensor with considerable accuracy, an approximate rectangle of 100 by 100 pixels is created at the desired coordinate and all the pixel values are averaged at that area. Once the average pixel values are received, it records deviation of each value from the expected values in each pixel and record the data in a binary file. Serialization technique is introduced in the code to avoid bottleneck in data flow. Once the error data is created and saved at an external file, in order to remove anomalies in depth measurement that error file is loaded and introduced to the execution flow so that each time a new depth frame arrives, those pre recorded error values are subtracted or added with real time depth values. Thus a considerable accuracy is achieved in measurement.

Another vital step in minimizing depth errors is introduction of time averaging. In order to remove spurious responses, an average value of pixels over the last 10 frames is calculated and stored in a buffer. At the time of processing a certain depth pixel, it is compared with time average of depth over the last 10 frames. If the deviation is within a tolerable limit, the data remains unaltered however a steep change in the depth received triggers the filter to bring the pixel value back into its average depth calculated over last few frames. Once the frame is filtered to a desired level, according to the depth of each pixel, an image is formed. Each pixel of the image so formed used to indicate distance between the sensor and the obstruction and hence is colored as per standard topographical convention. However due to limitation of the sensor itself any value below 400 millimeter is ignored and gets marked as unknown. Each time frame arrives from the sensor, the code behind checks if the frame is empty and if so happens it waits for the next frame to appear and discard the current frame. Otherwise, the frame is taken for further processing. Two vital methods involved in the process are creating a boundary in between the demarcation of different layers and mixing up two different colors with slowly changing gradient. Once the pixel array of the color image is finalized, the entire image is transferred to a bitmap source and the source property of the ultimate image to be projected is linked with the bitmap source so formed.

Once the final image is ready for projection and appears on the main window, all that is left out is remapping and rescaling the image. Rescaling assures that the image would not cross the boundaries of projection and intended interaction area. Whereas rescaling ascertain that peaks and crests are matching with the two dimensional color image so that it wraps precisely undulations of the surface. Failing to accomplish both above steps would result in wrong indication of depth and heights with unintended colors.

Code for the Simulation

Code used to filter unknown pixel surrounded by known pixels

```
for (int j = 640; j <= (depthFrame.PixelDataLength - 640); j++)
{
      int lineNum = j / 640;
      int elementNum = j % 640;

      if (((lineNum % 2) == 0) && ((elementNum % 2) == 0))
      {
            if ((depthData0[j] >> 3) == myKinect.DepthStream.UnknownDepth)
            {
                  if(((depthData0[j - 1] >> 3) != myKinect.DepthStream.UnknownDepth) &&
                     ((depthData0[j + 640] >> 3) != myKinect.DepthStream.UnknownDepth) &&
                     ((depthData0[j - 640] >> 3) != myKinect.DepthStream.UnknownDepth))
                     {
                         int depthValue = ((depthData0[j - 1] >> 3) + (depthData0[j + 640] >> 3) +
                               (depthData0[j - 640] >> 3)) / 3;
                         depthValue = depthValue << 3;
                         (depthData0[j]) = (short)depthValue;
                     }
               }
      }
}
```

Code used to store depth values of last 8 frames to be used for time averaging

```
using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
{
      if (depthFrame == null) return;

      frameCounter++;
      if (frameCounter >= 900) frameCounter = 0;

      if (depthData0 == null) depthData0 = new short[depthFrame.PixelDataLength];
      if (depthData1 == null) depthData1 = new short[depthFrame.PixelDataLength];
      if (depthData2 == null) depthData2 = new short[depthFrame.PixelDataLength];
      if (depthData3 == null) depthData3 = new short[depthFrame.PixelDataLength];
      if (depthData4 == null) depthData4 = new short[depthFrame.PixelDataLength];
      if (depthData5 == null) depthData5 = new short[depthFrame.PixelDataLength];
      if (depthData6 == null) depthData6 = new short[depthFrame.PixelDataLength];
      if (depthData7 == null) depthData7 = new short[depthFrame.PixelDataLength];

      if (depthColorImage == null) depthColorImage = new byte[depthFrame.PixelDataLength * 4];
      if (colorImage == null) colorImage = new int[depthFrame.PixelDataLength];

      if ((frameCounter % 8) == 0) depthFrame.CopyPixelDataTo(depthData0);
      if ((frameCounter % 8) == 1) depthFrame.CopyPixelDataTo(depthData1);
      if ((frameCounter % 8) == 2) depthFrame.CopyPixelDataTo(depthData2);
      if ((frameCounter % 8) == 3) depthFrame.CopyPixelDataTo(depthData3);
      if ((frameCounter % 8) == 4) depthFrame.CopyPixelDataTo(depthData4);
      if ((frameCounter % 8) == 5) depthFrame.CopyPixelDataTo(depthData5);
      if ((frameCounter % 8) == 6) depthFrame.CopyPixelDataTo(depthData6);
      if ((frameCounter % 8) == 7) depthFrame.CopyPixelDataTo(depthData7);
}
```

Propagation
A Journal of Science Communication

Code used to remove spurious values

```
for (int i = 0; i < depthFrame.PixelDataLength; i++)
{
        int depthValue = 0;
        if (depthData0 != null && depthData1 != null && depthData2 != null
        && depthData3 != null && depthData4 != null)
        {
                depthValue = (int)((depthData0[i] + depthData1[i] + depthData2[i]
                + depthData3[i] + depthData4[i] + depthData5[i] + depthData6[i] + depthData7[i]) / 8);
        }
        depthData0[i] = (short)depthValue;
}
```

Code used to calculate non linearity in measurement of depth considering normal height to be 1400 mm

```
    if (error == null) error = new short[depthFrame.PixelDataLength];

    for (int h = 0; h < depthFrame.PixelDataLength; h++)
    {
            int depth = depthData0[h] >> 3;
            error[h] = (short)(1400 - depth);
    }

    if (recordFlag == true)
    {
            storeCalibration.Record(error);
            storeCalibration.Stop();
            recordFlag = false;
    }
```

Code used to save calibration data generated out of nonlinearity of the sensor

```
    class errorRecorder
    {
        BinaryWriter writer;
        Stream recordStream;

        public void Start(Stream stream)
        {
            recordStream = stream;
            writer = new BinaryWriter(recordStream);
        }

        public void Record(short[] error)
        {
            if (writer == null)
                throw new Exception("You must call Start before calling Record");

            BinaryFormatter formatter = new BinaryFormatter();
            formatter.Serialize(writer.BaseStream, error);
        }

        public void Stop()
        {
            if (writer == null)
                throw new Exception("You must call Start before calling Stop");

            writer.Close();
            writer.Dispose();

            recordStream.Dispose();
            recordStream = null;
        }
    }
```

Code used to retrieve error saved at the time of calibration

```
class errorRetriever
{
	public short[] retrieveError(Stream stream, int len)
	{
		short[] retrievedError = null;

		BinaryReader reader = new BinaryReader(stream);
		BinaryFormatter formatter = new BinaryFormatter();

		retrievedError = new short[len];
		retrievedError = (short[])formatter.Deserialize(reader.BaseStream);

		reader.Dispose();
		return (retrievedError);
	}
}
```

Post filtration tinting of individual pixel to indicate depth

(following code shows the sample code for top most two levels)

```
if ((depthValue == myKinect.DepthStream.UnknownDepth) ||
(depthValue == myKinect.DepthStream.TooNearDepth) || (depthValue == myKinect.DepthStream.TooFarDepth))
{
	blue = 0;
	green = 0;
	red = 0;
	alpha = 255;
}
else
{
	/*** Level 14 ***/
	if (depthValue >= 1150 && depthValue < 1190)
	{
		blue = 255;
		green = 255;
		red = 255;
		alpha = 255;
	}

	/*** Level 13 ***/
	if (depthValue >= 1190 && depthValue < 1205)
	{
		seedR = 1205 - depthValue;
		seedR *= 7;
		seedR = (135 + seedR) & 255;

		seedG = 1205 - depthValue;
		seedG *= 7;
		seedG = (135 + seedG) & 255;

		seedB = 1205 - depthValue;
		seedB *= 7;
		seedB = (135 + seedB) & 255;

		blue = (byte)seedB;
		green = (byte)seedG;
		red = (byte)seedR;
		alpha = 255;
	}
}
```

3. Standard LCD projector mapped to match reliefs on the surface –

A standard projector equipped with 3 LCD technologies is utilized to project the image so formed on top of the material surface. The projector with an intensity of 4200 ANSI Lumen placed at a height of approximately 1750 millimeters is deployed typically for this application.

Placement of the projector used to be at an edge of the table so as to confirm the projection doesn't get hindered with the sensor placed right on top of the area of interest.

Algorithm involved

1. Wait for main application window to get loaded

2. Initialize kinect and enable depth frame with desired resolution

3. Initialize depth frame ready event and wait for a new depth frame to arrive

4. If depth frame is null, return else process depth frame

5. Copy data from data frame to short pixel array

6. Initiate filtration of depth data –

a. Perform pixel filtering

b. Remove distortions in depth measurement

c. Perform time averaging

7. Create new instance of color image pixel array of desired length

8. Pick up each pixel one after another by looping

9. If the pixel is undefined, don't process it further

10. If the pixel fall within the range of definition, relate the corresponding pixel in color image

11. Set the value of Red, Green, Blue and Alpha bytes individually as per definition

12. Once each pixel of the depth frame is processed, form the color image frame

13. Loop through all the pixels in the color image so formed and detect boundaries

14. Create line of demarcation between sea levels

15. Apply gradient brush in between levels if the same is above sea level.

16. Once final image is ready for projection process it for rescaling and remapping

17. Rescale the width and height of the color image to match the area of interest

18. Black out portions beyond the boundary of projection to hide unwanted interaction

19. Remap the color image by adjusting the X and Y coordinate of the image

20. Match undulations of the surface precisely with the projected image

21. Once rescaled and remapped, create a new instance of a writable bitmap

22. Set the source of the finally projected image to the writable bitmap

23. Copy pixels from color image to the writable bitmap to get it linked to the final image

24. Compile and run the code to see performance

25. Wait for 10 seconds for stabilization of depth data

26. Make the surface under observation as flat as possible

27. Initiate calibration and save calibration data

28. Create reliefs and observe if the changes are getting reflected

29. Load calibration data and see the difference in tracking

30. Once finished with all above steps, proceed for final setup

STANDARD DESKTOP PC RUNNING ON 64-BIT WINDOWS 7 EQUIPPED WITH C#.NET PACKAGE

ELSE

FILTER NOISE

1. Remove Nonlinearity in response of the filter itself
2. Time averaging of pixels to remove spurious response
3. Guessing values of unidentified pixels surrounded by known pixels

IF DEPTH FRAME IS NULL

DEPTH FRAME
640 X 480 PIXELS @ 30 FRS

FORMATION OF PIXEL ARRAYS BY TINTING INDIVIDUAL PIXELS AS PER UNDULATION OF SURFACE

COPYING PIXELS OF THE ARRAY TO A BITMAP SOURCE

USB 2.0 OR ABOVE

ASSIGNING SOURCE OF THE PROJECTED IMAGE TO THE BITMAP SOURCE

REMAPPING THE IMAGE AS PER WIDTH AND HEIGHT OF INTERACTION SPACE

COMPARISON WITH REFERENCE PATTERN AND DEPTH CALCULATION

RESCALING THE IMAGE TO MATCH CRESTS AND PEAKS OVER THE SURFACE PRECISELY

MICROSOFT KINECT FOR WINDOWS VERTICALLY MOUNTED

CEILING MOUNTED PROJECTOR

IR PROJECTOR

IR CAMERA

RGB CAMERA

PROJECTION OF THE IMAGE SO FORMED

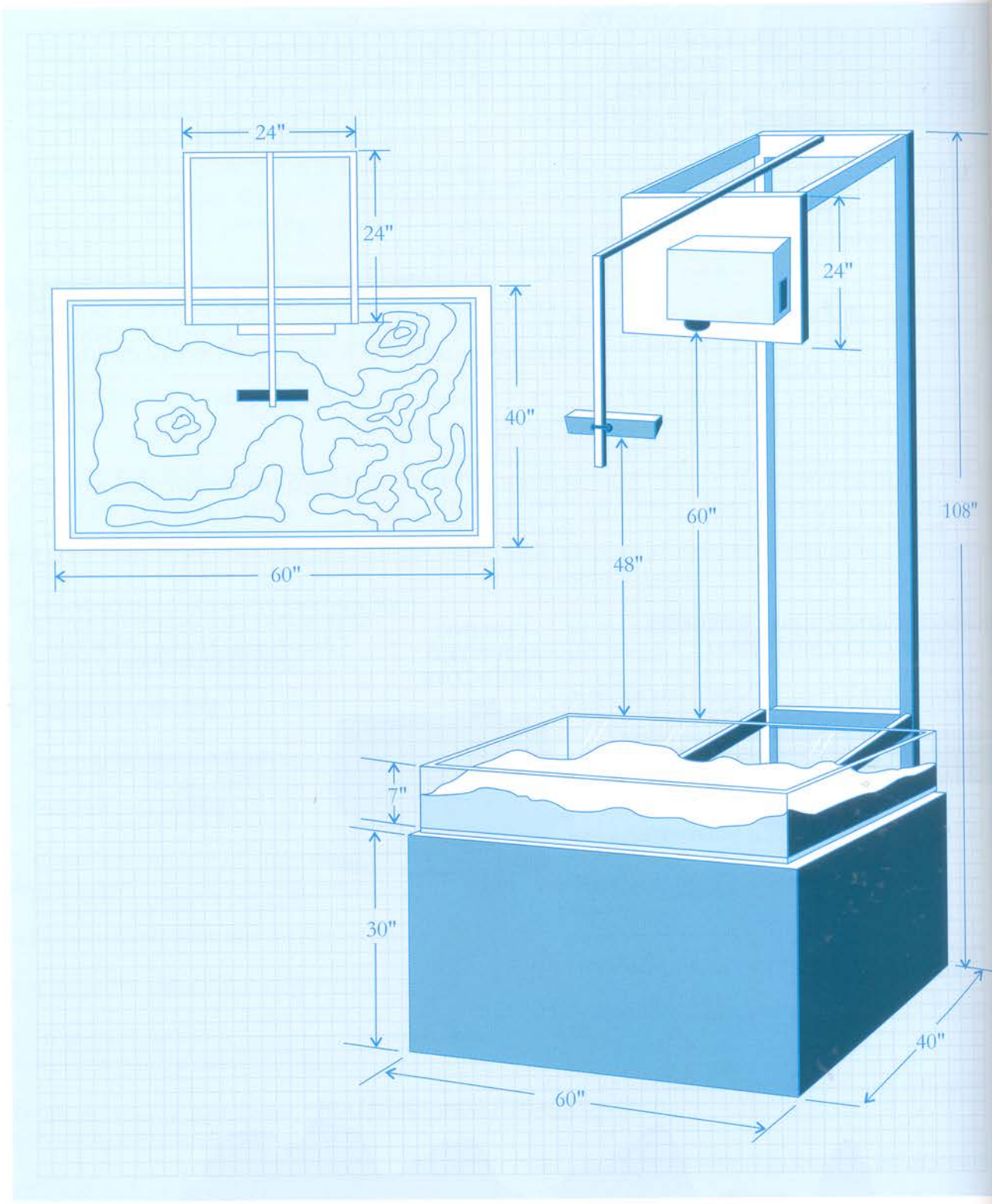
PSEUDO RANDOM PATTERN OF IR DOTS

SNAP OF INCIDENT PATTERN

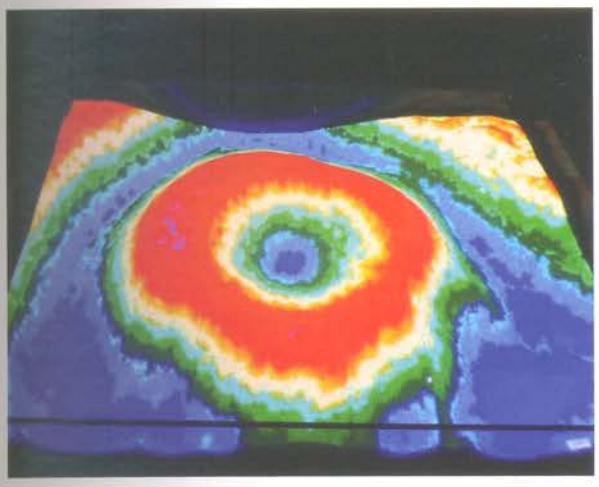
SURFACE UNDER CONSIDERATION

Propagation
A Journal of Science Communication

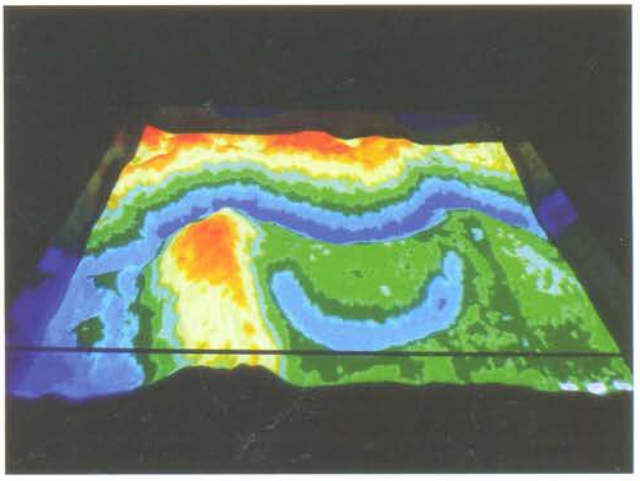
Schematic Drawing of the Setup –



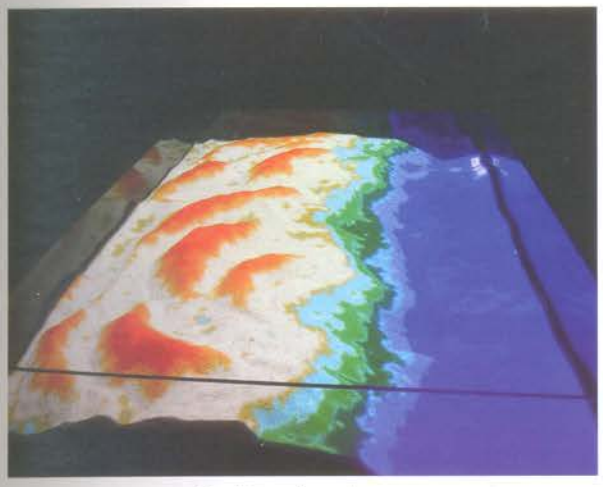
A few common topographical land forms simulated on the surface



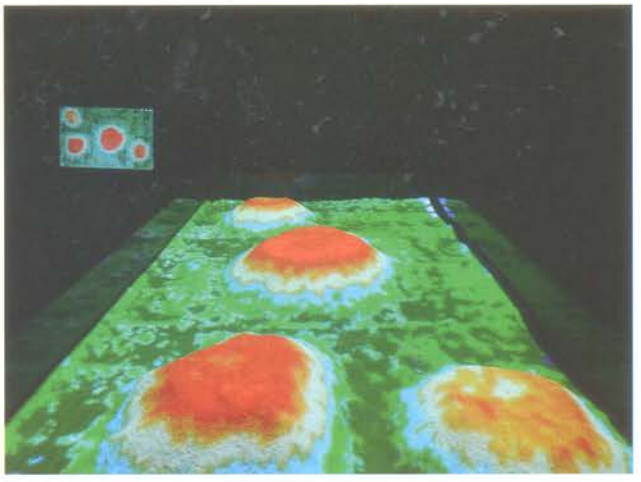
Lake indide an Island



Flow path of a river and formation of horse shoelake



Sand dunes formation



Formation of Plato

Propagation
A Journal of Science Communication

Conclusion

Initially it may look like a fun tech demonstration; however intended use may cover serious applications in the field of simulation. Starting with rain water flow path simulation to landscaping and determining unstable land forms prone to erosion and landslides may well be understood with such system. It may too play a vital role in making people understand formation of different geographical landscapes. Above all, its application in the field of non formal education in science centre and museums would be boundless.

The authors owe their gratitude to the Director NSCM, who compelled us to attempt this project. Not with standing the initial problems, that we faced in achieving the desired result, the final out come was truly remarkable, which we witnessed at the Indian Science Congress 2015, where this exhibit was first displayed.

Acknowledgements

1. Start here, Learn the Kinect API – by Rob Miles

2. Programming with Kinect for windows SDK – by David Catuhe

3. Smoothing Kinect depth frame in real time – an article by Karl Sanford

4. Hand Detection and positioning based on Kinect depth image – by Van Bang Le, A.T. Naguyen, Yu Zhu

5. Recovering missing depth information from Kinect depth image – by Abdul Dakkak, Amar Hussain

Arnab Chatterjee
Curator C
Nehru Science Centre, Mumbai

Shrikant P. Pathak
Curator F
Nehru Science Centre, Mumbai