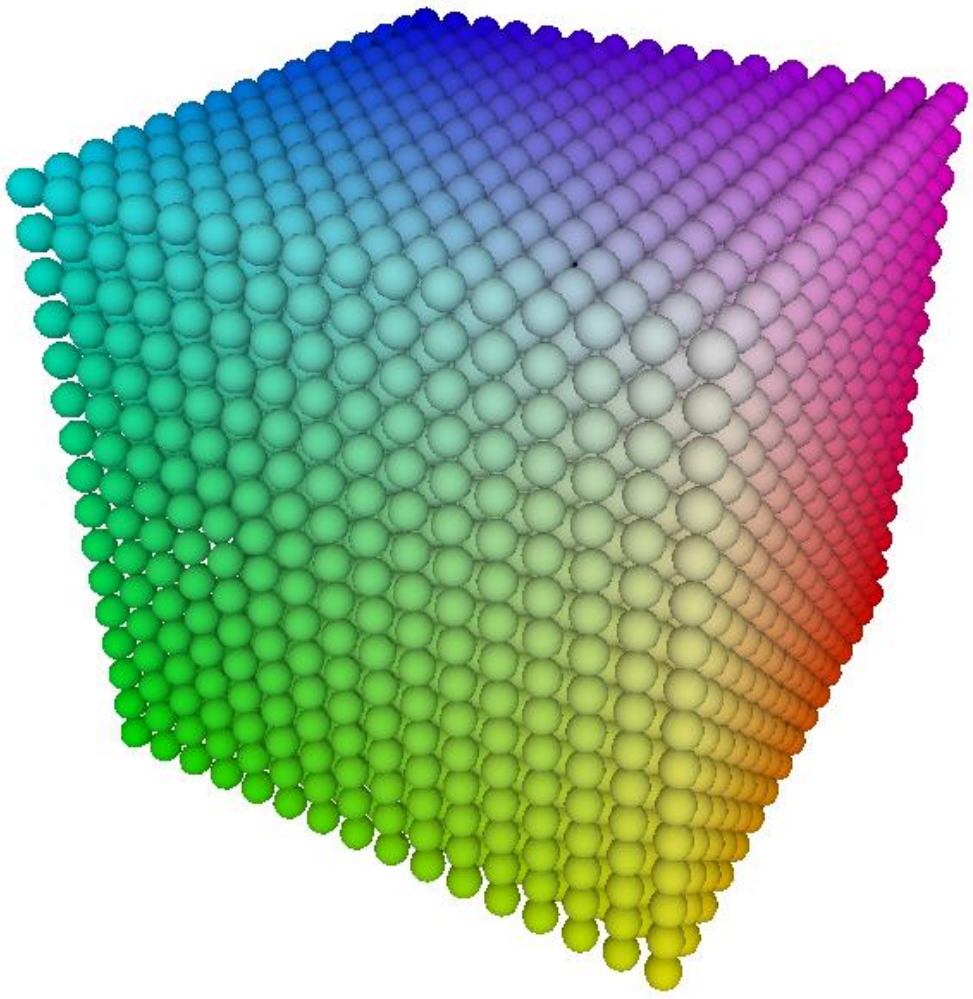


CubeSense

User's Guide

V1.0.704.1

Table of Contents

1 - Introduction

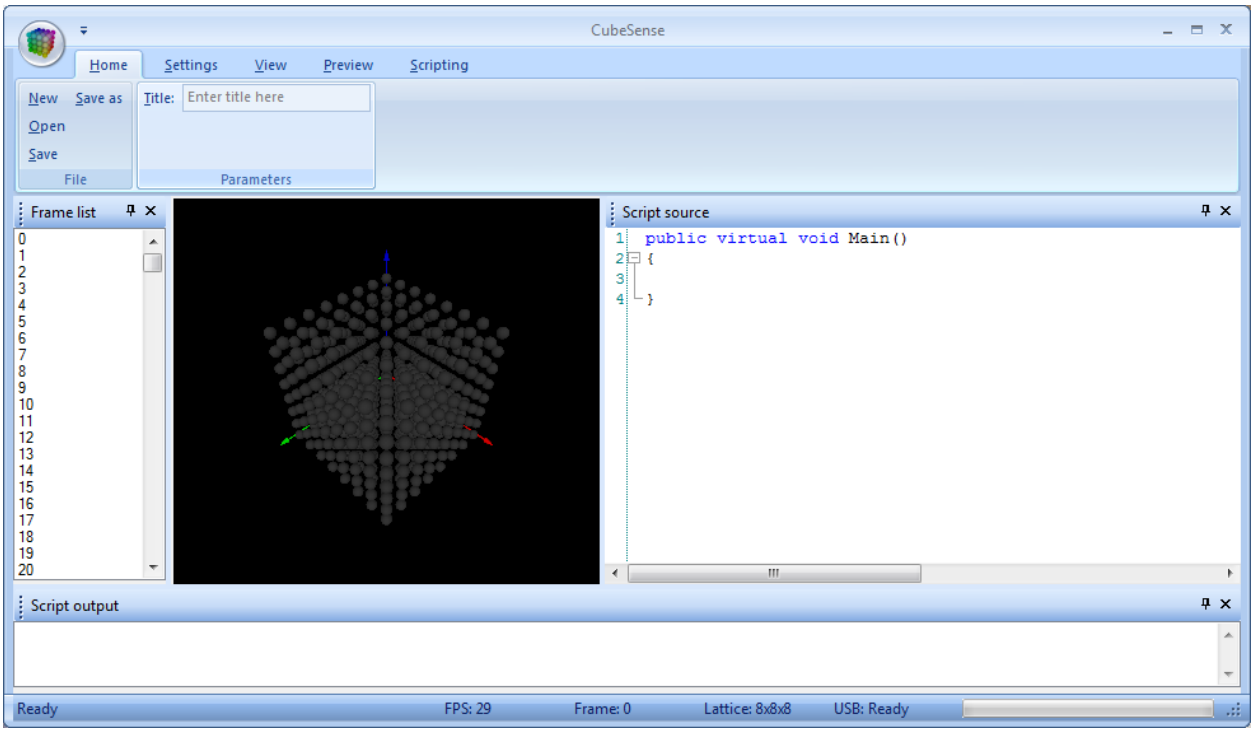
CubeSense is a free PC application created by Lumisense (www.lumisense.com). At the moment, it runs only under Windows OS. Its primary purpose is to be used as a tool in creating content to show on the Lumisense eightCubed. This application allows creation of raw 3D RGB video files, which can later be played back on an eightCubed system. The format of created files (described later on) is very simple, which means the application can be used as 3D context rendering tool for basically any relatively low resolution 3D graphics display. Additionally, CubeSense allows real time preview of rendered animations when the eightCubed is connected to a PC via USB.

As mentioned above, the CubeSense application is free to download at the Lumisense website. We encourage our customers (and anyone willing) to explore CubeSense's possibilities by creating 3D animations.

2 - Installation

Download CubeSense.rar file from www.lumisense.com. After extracting it, run setup.exe. It will take you through the process of installing the application. When installation is complete, CubeSense can be started with a shortcut created on the desktop or in start menu. It can also be launched directly from installation directory.

3 - User Interface



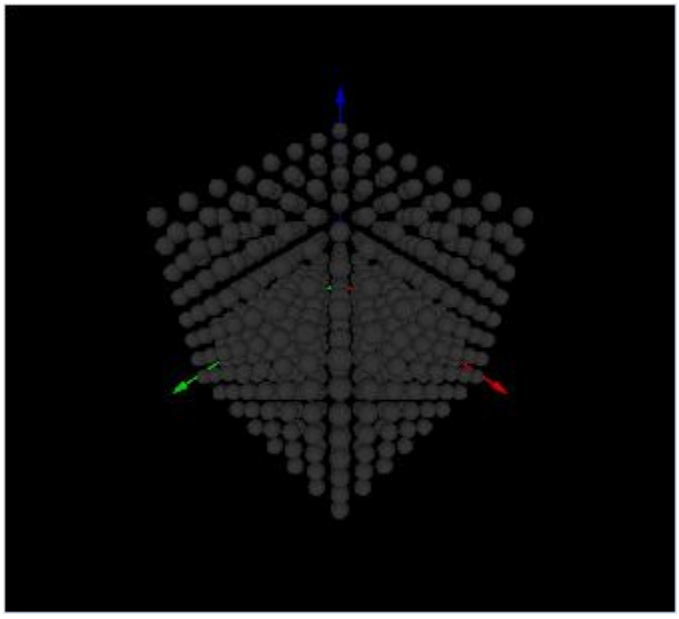
After the CubeSense application is launched, a main application window should appear. By default, it consists of the following elements:

- *Main visualization area* – this is where the lattice will be displayed and animations can be previewed.

- *Frame list* – displays all frames in current animation file, allowing a user to select any of them for preview.
- *Script output* – output window used by scripting engine. It displays compilation results as well as any additional information by user script.
- *Script source* – this is where script text is being written before it's executed.

Additionally, there is a ribbon bar at the top of the application window. The following subsections will discuss each of these elements in greater details.

3.1 - Main Visualization Area



This is where the lattice and selected frame are being displayed.

Rotate

To rotate the lattice in 3D space, move the mouse while holding the right mouse button.

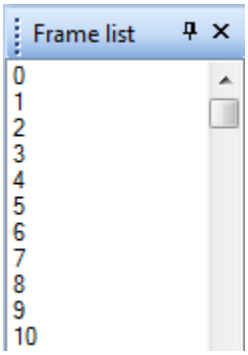
Continuous Rotation

If the right mouse button is released while the pointer is in motion, the lattice will continue rotating in the same direction as it was before releasing the mouse button.

Zoom

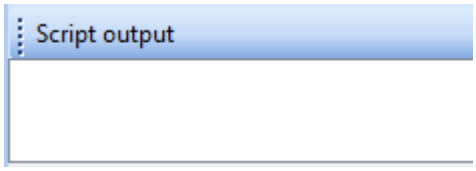
Rotating mouse wheel while holding the same right mouse button allows a user to zoom in and out.

3.2 - Frame List Area



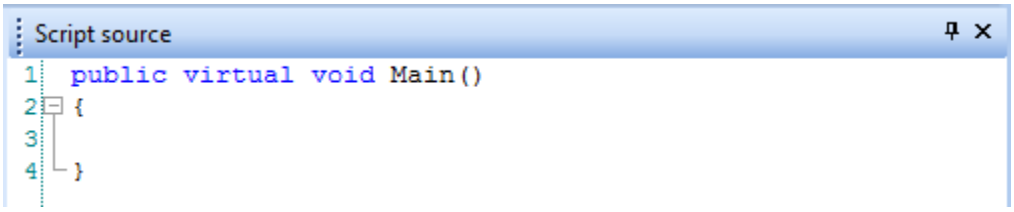
A frame can be activated by clicking on its frame list record. The active frame will be displayed in the Main Visualization Area. Several sequential frames may be selected at the same time, although only one of them will actually be displayed. This is a useful feature for using the Repeat feature in Playback Mode.

3.3 - Script Output Area



This area shows output of the scripting engine. If there were errors during script compilation, they will be shown in this window. Clicking on any of them will immediately bring the cursor to the target line within the Script Source window.

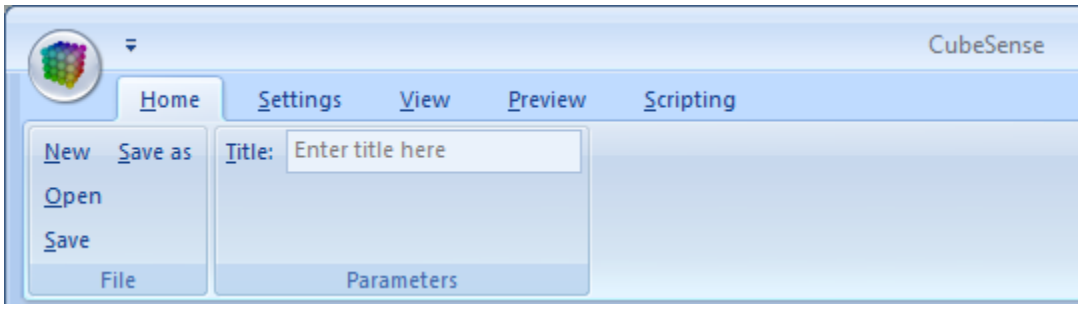
3.4 - Script Source Area



In this window you can write your script code. To compile and execute your script, click on the Run command, in the Scripting page of the ribbon bar. Alternatively hit F5.

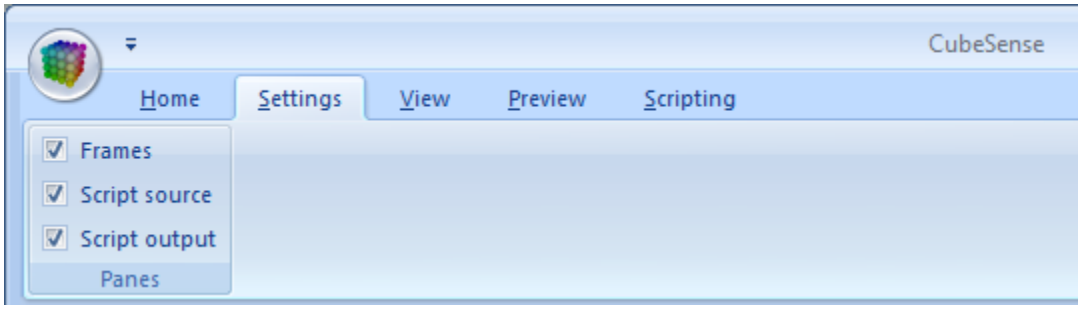
3.5 - Ribbon Bar

Home



The *Home* page of the ribbon bar has the following commands: *New*, *Open*, *Save*, *Save as*. When you click *New* a small window opens up. It allows you to specify the lattice characteristics for a new animation. After choosing the number of frames in the new animation, lattice shape and size, then clicking the *Ok* button, a new animation will be created with all frames initially blank. The rest of the just mentioned commands are self explanatory – they allow you to save and open CubeSense animation files with *.eca extension. On *Home* page you may also enter animation title which will be displayed on eightCubed LCD when the animation is being played.

Settings



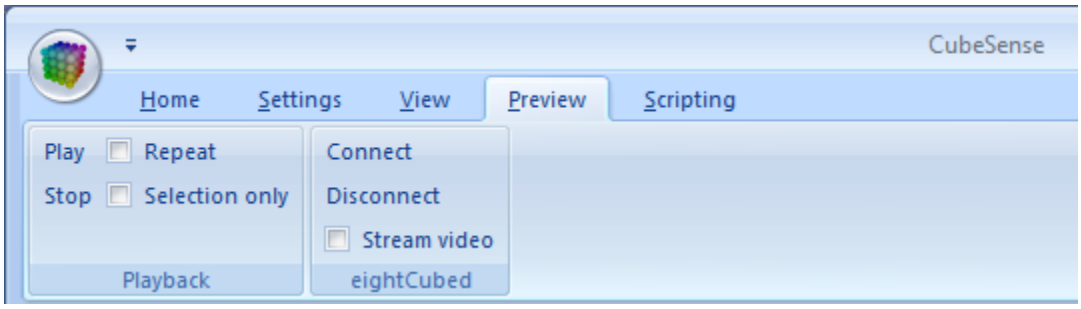
The *Settings* page allows you to hide and show any of three Areas: *Frame List, Script Output, and Script Source*.

View

In the V*iew* page you can choose between several quick presets of lattice orientation: X-Z view, X-Y view, Y-Z view and default view. It also lets you choose whether you want lattice and axis to be visible or not. In *lattice* group of *view* page you can choose LED size and their spacing. By default, in the

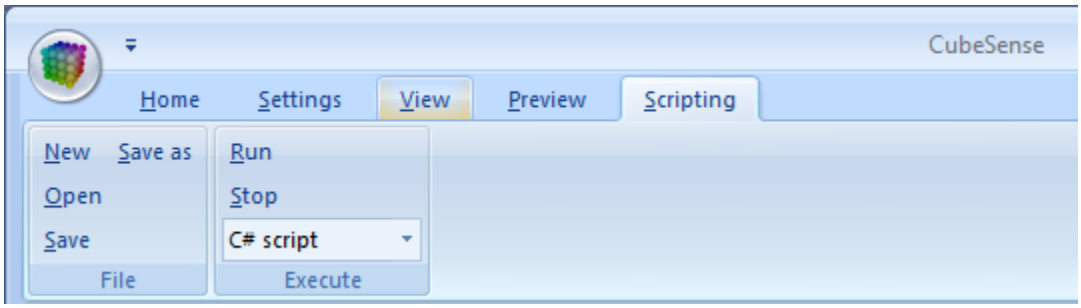
eightCubed kit, LED size is equal to 10mm and spacing is 36mm. Additionally, you can choose how you want black LEDs to be shown. They can either be black, dark grey or invisible. And finally you can select quality for lattice preview. Drawing that many of the spheres (depending on your selected lattice size) may be resource consuming, so choose render quality to best suit your needs. For your reference FPS count is shown at the bottom in the status bar.

Preview



The *Preview* page has buttons to start and stop automatic switching between frames in current animation (aka. playback). If several frames are selected in frame list window and *Selection only* checkbox is checked only those frames will be played back. Checking *Repeat* checkbox makes application repeat playback one it has reached last frame of animation or selection (depending on *Selection only* check box). If you have eightCubed kit connected to you PC click *Connect* button to connect to the kit. After successful connection is established you can check *Stream video* checkbox and your real cube will show exactly what's on the screen.

Scripting



The *Scripting* page allows you to perform all the default file operations with the script code. Clicking *Run* compiles your script and shows errors if there were any. If not it will run the script. You can also select one of two ways you script is going to be treated – either C# (C sharp) or VB (visual basic) script. This selection is important when you click Run. It is also used by application to highlight syntax properly. Clicking Stop will send a message to your script that it's time to stop. For more details on how this function has to be used look in Scripting section of this document.

4 - Scripting

In order to create animations you will have to write some scripts. Drawing animations frame by frame, pixel by pixel is definitely not the way to go. Those of you who have at least very basic knowledge in any kind of programming should find scripting fairly easy. Others should be able to learn it quickly form several examples installed along with CubeSense application.

As mentioned before, two scripting languages can be used – VB and C#. Further in this guide we will be using latter one.

To begin with, any of your scripts must contain Main function which is declared as follows: public **virtual void Main();** It is actually created for you automatically when you click *New* in *Scripting* page. All you have to do then is to write something in between the { and } brackets. If you like you can create additional functions to be called from **Main()** function. **Main()** function is the function that scripting engine will try to call once the script has been successfully compiled.

To interact with the lattice and CubeSense environment, namespace **cs** (short for CubeSense) is available. For example to draw a voxel (volumetric pixel, or simply pixel in 3D world) you have to write the following line inside of the main function:

```
cs.drawVoxel(f,x,y,z,r,g,b);
```

Where **f** is frame number, **x,y,z** are coordinates of the voxel, and **r,g,b** are color values for red, green and blue color channels respectively. After replacing all these parameters with fixed values or assigning these variables with certain numbers, you can run the script and see single voxel of your choice be painted.

Painting single voxels is not much fun, therefore common cycle instruction **for** can be used to handle many voxels. The next few lines of C# script code would create a rainbow pattern in frame 0:

```
for(x=0;x<cs.getSizeX();x++)
    for(y=0;y<cs.getSizeY();y++)
        for(z=0;z<cs.getSizeZ();z++)
            cs.drawVoxel(0,x,y,z,256/cs.getSizeX()*x,256/cs.getSizeY()*y,256/cs.getSizeZ()*z);
```

As you can see, several rather self explanatory functions from the **cs** name space have been used to find the size of lattice so that the script would not depend on it.

If you wish to be able to terminate your script at any time add to it this line:

```
if (cs.checkTermination())return;
```

Function **cs.checkTermination()** will return **true** if *Stop* button has been pressed in *Scripting* page, thus **Main()** function will return.

4.1 - cs Namespace Functions

To provide more scripting power, new functions will be added in the near future.

cs.drawVoxel(f, x, y, z, r, g, b);

Draws single voxel having color **r, g, b** in frame **f** at position **x, y, z**.

cs.drawCube(f, x1, y1, z1, x2, y2, z2, r, g, b);

Draws a filled cube having color **r, g, b** in frame **f**.

x1, y1, z1 and **x2, y2, z2** specifies two cube's corners

cs.getFrameCount();

Returns integer number of frames in current animation.

cs.getSizeX();

Returns integer lattice size in X direction in current animation.

cs.getSizeY();

Returns integer lattice size in Y direction in current animation.

cs.getSizeZ();

Returns integer lattice size in Z direction in current animation.

cs.getColorR(f, x, y, z);

Returns integer value of the red color channel of the voxel located at position **x, y, z** in frame **f**

cs.getColorG(f, x, y, z);

Returns integer value of the green color channel of the voxel located at position **x, y, z** in frame **f**

cs.getColorB(f, x, y, z);

Returns integer value of the blue color channel of the voxel located at position **x, y, z** in frame **f**

cs.checkTermination();

Returns boolean True if Stop button has been clicked in Scripting page.

cs.showProgress(p);

Displays integer value **p** in progress bar at the bottom or CubeSense window.

Value 0 means progress bar is empty, and value 100 means progress bar is full.

4.2 - Function Return Codes

All cs Namespace functions that receive an **f, x, y** or **z** parameter can potentially return an error code (a negative integer value) indicating that the invoking call failed due to an invalid argument being passed. Error codes returned in this manner are associated with specific causes of failure as follows:

Return Code	Description
-1	Wrong size – an attempt was made to define a lattice size exceeding the maximum limitations of the program
-2	Wrong frame – the value of the frame (**f**) call argument is outside the range of frames defined in the Frames List
-3	Wrong voxel – the value of a voxel position (**x, y** or **z**) call argument is outside the defined dimensions of the lattice. Also applies to **x1, x2, y1, y2, z1** and **z2** values

A script author can programmatically determine the success/failure of most **cs** function calls by invoking them in a manner similar to that shown in the following C# script example:

```
int rc; // Return Code from Function Call
rc=cs.drawVoxel(f, x, y, z, r, g, b);
if (rc<0)
{
// drawVoxel call failed - diagnose specific cause by evaluating the value of rc
…
}
```

5 - File Formats

When an animation is created using scripting in CubeSense, it can be saved to disk in ***.eca** format. Information in this section is useful if you wish to read/write ***.eca** files yourself.

At the beginning of each ***.eca** file is a header section. The structure of the header is shown in the following table:

Address	Length	Description
0x0000	0x0002	always 0x4C73
0x0002	0x0001	type of file. Contains value of 1 in *.eca files created by CubeSense.
0x0003	0x0002	RTA animation (Not used in *.eca files created by CubeSense)
0x0005	0x0004	Number of frames (for "frame stream" file)
0x0009	0x0003	Lattice size (1st byte - x, 2nd byte - y, 3rd byte - z)
0x000C	0x0020	Animation title (upto 20 visible chars+'/0')

After the header, starting at 0x0100 address in *.eca file goes actual 3D video data. Each frame takes up 3*sizeX*sizeY*sizeZ (where sizeX, sizeY and sizeZ are lattice size parameters) bytes. First third of the frame is for red color channel's data. Position of voxel in this first third when its coordinates are known can be found by the following equation: ((z*sizeY)+y)*sizeX+x. Following red channel goes green channel and finally blue channel. Next is the next frame with the same data structure.