First Block Checkoff

James Wilcock

January 28, 2022

Description of Block

This block is the code is written for controlling of the 5x5x7 led cube. As our cube has not yet been constructed fully in order to simulate the code for controlling the leds using the 3d modeling program Blender to display what the cube will approximately look like when operational. As such the code is not exactly what will be written for the ESP32 microcontroller but the general algorithm will remain the same. This code is written in python and is targeting the blender api in order to place virtual leds and wires for dramatic effect.

Image of Simulation

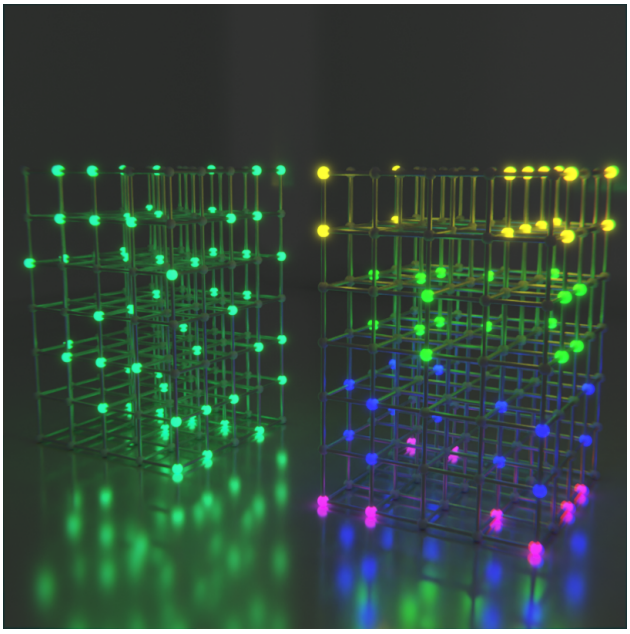
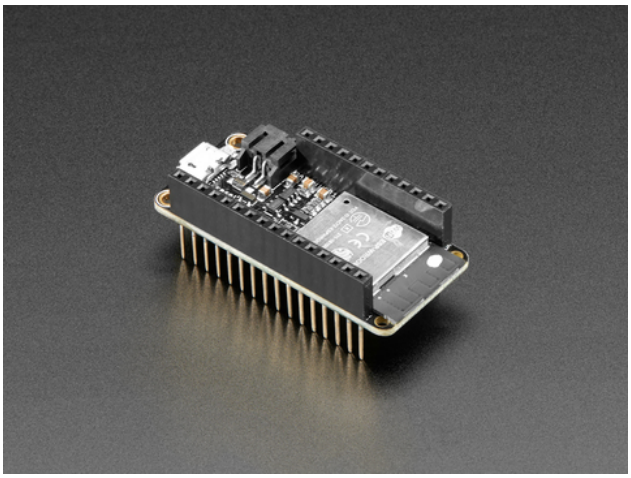


Image of ESP32 to be used



Simulation code

```
import bpy, bmesh
import math
import numpy as np
from numpy.random import default_rng
C = bpy.context
D = bpy.data
LED_RADIUS = .10 # radius of spheres
LED_BRIGHTNESS = 8
CYL_RADIUS = .04 # radius of "wires"
CYL_VERTICES = 6
LED_SEGMENTS = 12
DEFAULT_LEDCOLOR = (1,.5,1,1)
raincolor = (0,143/255,17/255,1)
RAINLENGTH = 3
width = 5 # x
length = 5 # y
height = 7 # z

#Deletes the led cube that was previously generated
def deletecubeandmaterials():
    for material in bpy.data.materials:
        if "led" in material.name:
            #material.user_clear()
            bpy.data.materials.remove(material)
    for mesh in bpy.data.meshes:
        if "Icosphere" in mesh.name or "Sphere" in mesh.name or "Cyl" in mesh.name:
            bpy.data.meshes.remove(mesh)
    print("Deleted led materials and models")

#makes a cylinder between the points (x1,y1,z1) and (x2,y2,z2)
def cylinder_between(x1, y1, z1, x2, y2, z2, r):
  dx = x2 - x1
  dy = y2 - y1
  dz = z2 - z1
```

```
  dist = math.sqrt(dx**2 + dy**2 + dz**2)

  bpy.ops.mesh.primitive_cylinder_add(
      vertices = CYL_VERTICES,
      radius = r,
      depth = dist,
      location = (dx/2 + x1, dy/2 + y1, dz/2 + z1)
  )

  phi = math.atan2(dy, dx)
  theta = math.acos(dz/dist)

  bpy.context.object.rotation_euler[1] = theta
  bpy.context.object.rotation_euler[2] = phi


  wiremat = D.materials["Wire"]


  ob = bpy.context.active_object

  ob.active_material = wiremat
  """
  mat = bpy.data.materials.get("Material")

  if mat is None:
      mat = bpy.data.materials.new(name="wire")

  if ob.data.materials:
      ob.data.materials[0]=mat
  else:
      ob.data.materials.append(mat)
  """

#adds material to the led with a emission color
#controlled by the color input
def addledmaterial(color,brightness):
    ob = C.active_object

    mat = bpy.data.materials.get("Material")

    if mat is None:
      mat = bpy.data.materials.new(name="led")

    if ob.data.materials:
      ob.data.materials[0]=mat
    else:
      ob.data.materials.append(mat)

    mymat = ob.active_material.name
    #print(mymat)
    mat = bpy.data.materials.get(mymat)

    mat.use_nodes = True
```

```
    nodes = mat.node_tree.nodes

    nodes["Principled BSDF"].inputs[19].default_value = color
    nodes["Principled BSDF"].inputs[20].default_value = brightness # brightness of led
    #bpy.data.materials["led"].node_tree.nodes["Principled BSDF"].inputs[19].default_value = (0.661184,

#gets the respective rainbow color to return for
#a z level
def getrainbow(z):
    color = []
    if (z == 0):
        color = [148, 0, 211]
    elif (z==1):
        color = [75, 0, 130]
    elif (z==2):
        color = [0, 0, 255]
    elif (z==3):
        color = [0, 255, 0]
    elif (z==4):
        color = [255, 255, 0]
    elif (z==5):
        color = [255, 127, 0]
    elif (z==6):
        color = [255, 0 , 0]
    color.append(255)
    color = [x/255 for x in color]
    return color

#place leds and the 'bars' ie the wires
#in real life
def placeall(spherelist):
    for i in range(width):
        spherelist.append([])
        for j in range(length):
            spherelist[i].append([])
            for k in range(height):
                spherelist[i][j].append([])
    count = 0
    for z in range(height):
        for y in range(length):
            for x in range(width):
                print("Placing led ", count)
                bpy.ops.mesh.primitive_ico_sphere_add(subdivisions = 1,radius=LED_RADIUS, enter_editmod
                ,location=(x,y,z), scale=(1,1,1))
                spherelist[x][y][z].append(bpy.ops.object)
                color = DEFAULT_LEDCOLOR
                addledmaterial(color,LED_BRIGHTNESS)
                count += 1
    print("Placed leds")

    for x in range(width):
        for y in range(length):
            cylinder_between(x,y,0,x,y,height-1,CYL_RADIUS)
            for z in range(height):
```

```
                if y == 0:
                    cylinder_between(x,0,z,x,width-1,z,CYL_RADIUS)
                if x == 0:
                    cylinder_between(0,y,z,width-1,y,z,CYL_RADIUS)
    print("Placed bars")


def rainbowanimate(spherelist):
    frame_num = 0
    znum = 0
    for count in range(50):
        for z in range(height):
            for y in range(length):
                for x in range(width):
                    shadnum = str(x*1+y*(width)+z*(width*length)).zfill(3)
                    if shadnum != '000':
                        mat = bpy.data.materials.get('led.'+shadnum)
                    else:
                        mat = bpy.data.materials.get('led')
                    nodes = mat.node_tree.nodes
                    #input[19] is the emission color
                    #input[20] is the emissions strength, aka brightness
                    nodes["Principled BSDF"].inputs[19].default_value = getrainbow((znum+z)%7)
                    nodes["Principled BSDF"].inputs[20].default_value = LED_BRIGHTNESS
                    nodes["Principled BSDF"].inputs[19].keyframe_insert("default_value",frame=frame_num
        znum += 1

        frame_num += 5
    print(f"Finished keyframes to frame {frame_num-20}")

#keyframes the led color in order to allow for
#animations
def keyframeleds(x,y,z,frame_num,color,highestnum):
    if highestnum > 0:
        shadnum = str(highestnum+1 + x*1+y*(width)+z*(width*length)).zfill(3)
    else:
        shadnum = str(highestnum + x*1+y*(width)+z*(width*length)).zfill(3)
    if shadnum != '000':
        mat = bpy.data.materials.get('led.'+shadnum)
    else:
        mat = bpy.data.materials.get('led')
    nodes = mat.node_tree.nodes
    nodes["Principled BSDF"].inputs[19].default_value = color #(1,.4,1,1)
    nodes["Principled BSDF"].inputs[20].default_value = LED_BRIGHTNESS
    nodes["Principled BSDF"].inputs[19].keyframe_insert("default_value",frame=frame_num)

#generates rain effect, choosing iterations number
#of random spots on the top of the cube to
#drop through
def rain_effect(iterations,ar,color = raincolor):
    rng = default_rng()
    randlist = []
    for x in range(iterations):
        randx = rng.integers(width)
```

```
        randy = rng.integers(length)
        randlist.append([randx,randy])
        ar[int(randx)][int(randy)][height-1] = color
    return ar,randlist

#shifts the color array down, if moving
#from bottom most plane down it gets removed
def shiftarray(ar):
    ar = np.roll(ar,-1,axis=2)
    ar[:,:,height-1:] = [0,0,0,0]
    return ar

#does the rain effect on the led array,
#generating the rain drops
def dorain(ledarray,its,color = raincolor):
    randlist = []
    ledarray = shiftarray(ledarray)
    ledarray,randlist = rain_effect(its,ledarray,color)
    return ledarray,randlist

#sets the keyframes constant so that colors
#and other values change instantly
def setkeyframesconstant():
    for obj in bpy.data.materials:
        if obj.node_tree.animation_data:
            fc = obj.node_tree.animation_data.action.fcurves
            for fcurve in fc:
                for kf in fcurve.keyframe_points:
                    kf.interpolation = 'CONSTANT'

#Creates a cube in center of array,
#had to be hand coded.
def docube(colorlist,frame_num):
    """
    1,1,2
    1,2,2
    1,3,2
    2,3,2
    3,3,2
    3,2,2
    3,1,2
    2,1,2
    these points plus all of these points +1 and +2
    """
    points = [(1,1,2),(1,2,2),(1,3,2),(2,3,2),(3,3,2),(3,2,2),(3,1,2),(2,1,2)]
    for point in points:
        for x in range(3):
            mypoint = (point[0],point[1],point[2]+x)
            if mypoint not in [(2,1,3),(3,2,3),(1,2,3),(2,3,3)]:
                colorlist[point[0],point[1],point[2]+x] = DEFAULT_LEDCOLOR

    if frame_num % 20 == 0:
        print(frame_num)
        print(colorlist[1])
```

```
        colorlist = np.rot90(colorlist)
        print(colorlist[1])

    return colorlist
#calls the rain function with the only difference being the color passed
# in is the rainbow
def dorainbowrain(dripcount, count,frame_num,colorlist,randlist):
    if RAINLENGTH > count >= 0 and frame_num % 10 == 0 and frame_num > 0:
        colorlist = shiftarray(colorlist)
        for rand in randlist:
            #print(colorlist[int(rand[0])][int(rand[1])][height-1])
            colorlist[int(rand[0])][int(rand[1])][height-1] = getrainbow(dripcount % height)
            #print(colorlist[int(rand[0])][int(rand[1])][height-1])
        count += 1
    elif frame_num % 10 == 0:
        if frame_num == 0:
            colorlist,randlist = dorain(colorlist,10, getrainbow(dripcount % height))
        else:
            colorlist,randlist = dorain(colorlist,10, getrainbow((dripcount + 1) % height))
        count = 0
        dripcount += 1
    return colorlist,randlist,count, dripcount

#takes care of single colored rain, using the raincolor global
def dosinglecolorrain(count,frame_num,colorlist,randlist):
    if RAINLENGTH > count >= 0 and frame_num % 10 == 0 and frame_num > 0:
        colorlist = shiftarray(colorlist)
        for rand in randlist:
            #print(colorlist[int(rand[0])][int(rand[1])][height-1])
            colorlist[int(rand[0])][int(rand[1])][height-1] = raincolor
            #print(colorlist[int(rand[0])][int(rand[1])][height-1])
        count += 1
    elif frame_num % 10 == 0:
        colorlist,randlist = dorain(colorlist,10)
        count = 0

    return colorlist,randlist,count

# catch all function to animate based on the type input
# calls all functions needed to keyframe etc.
def animateany(spherelist,type, highestsphere):
    frame_num = 0
    count = 0
    randlist = []
    colorlist = np.zeros((width,length,height,4)) #big list of colors
    dripcount = 0
    for frame_num in range(0,500,10):
        for z in range(height):
            for y in range(length):
                for x in range(width):
                    color = colorlist[x,y,z]
                    keyframeleds(x,y,z,frame_num,color, highestsphere)
        if type == "rain":
            colorlist,randlist,count = dosinglecolorrain(count,frame_num,colorlist,randlist)
```

```
        if type == "rainbowrain":
            colorlist, randlist, count, dripcount = dorainbowrain(dripcount, count,frame_num,colorlist,
        count += 1
        if type == "cube":
            colorlist = docube(colorlist,frame_num)

    if type == 'rain' or type == 'rainbowrain':
        setkeyframesconstant()

    print(f"Finished keyframes to frame {frame_num-20}")

#initializes a 4d array of sphere colors
def makespherelist():
    spherelist = []
    for i in range(width):
        spherelist.append([])
        for j in range(length):
            spherelist[i].append([])
            for k in range(height):
                spherelist[i][j].append([])
    return spherelist

#gets the highest sphere number to control for if
# making multiple cubes
def gethighestsphere():
    name = ""
    highestnum = 0
    for mesh in bpy.data.meshes:
        if "Sphere" in mesh.name:
            name = mesh.name
            print(name)
    if name:
        highestnum = int(name.split('.')[1])
    print(highestnum)
    return highestnum

if __name__ == '__main__':
    deletecubeandmaterials()
    highestsph = gethighestsphere()
    spherelist = makespherelist()
    placeall(spherelist)
    #rainbowanimate(spherelist)
    animateany(spherelist,"rain",highestsph)
    print("Done")



"""
spinning 3d shape would be cool
also want to do 3d cellular automata such as
conways game of life
```

```
"""
```