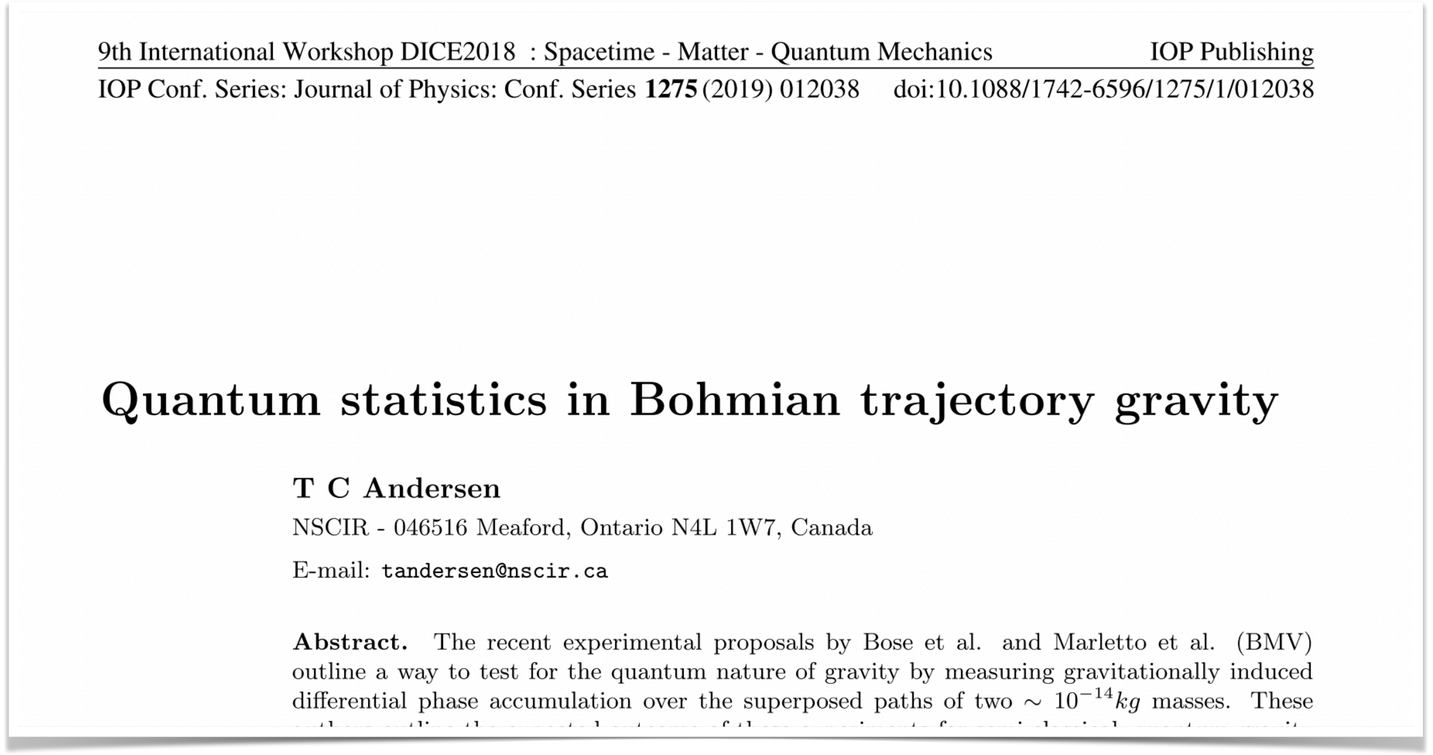
Cosmic Ray App - iOS

Detection and graphics algorithms.

Tom Andersen nSCIr.ca 2021

Tom Andersen

- PhD on Sudbury Neutrino Observatory (SNO).
- SNO - Low level radiation counting



9th International Workshop DICE2018 : Spacetime - Matter - Quantum Mechanics IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **1275** (2019) 012038 doi:10.1088/1742-6596/1275/1/012038

Quantum statistics in Bohmian trajectory gravity

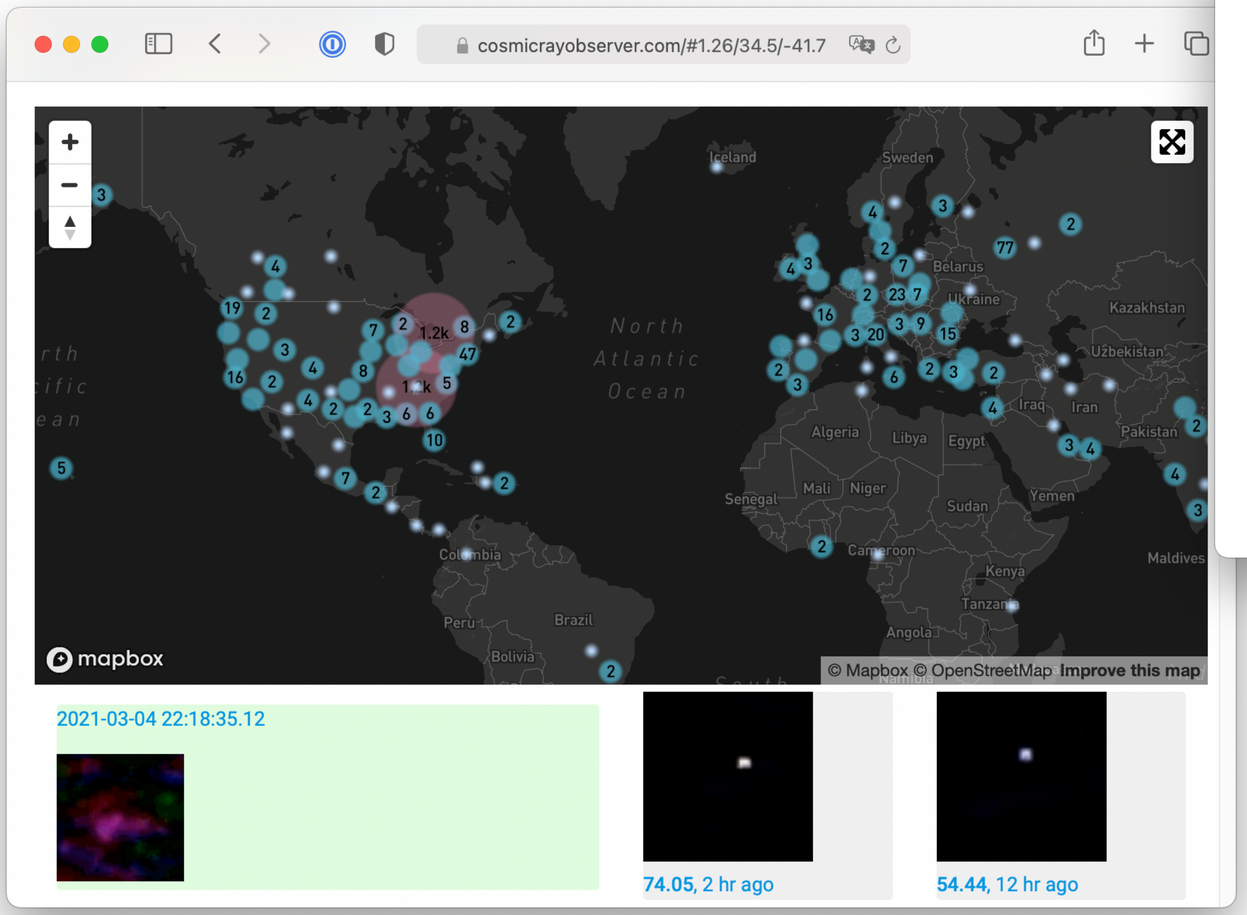
T C Andersen
NSCIR - 046516 Meaford, Ontario N4L 1W7, Canada
E-mail: tandersen@nscir.ca

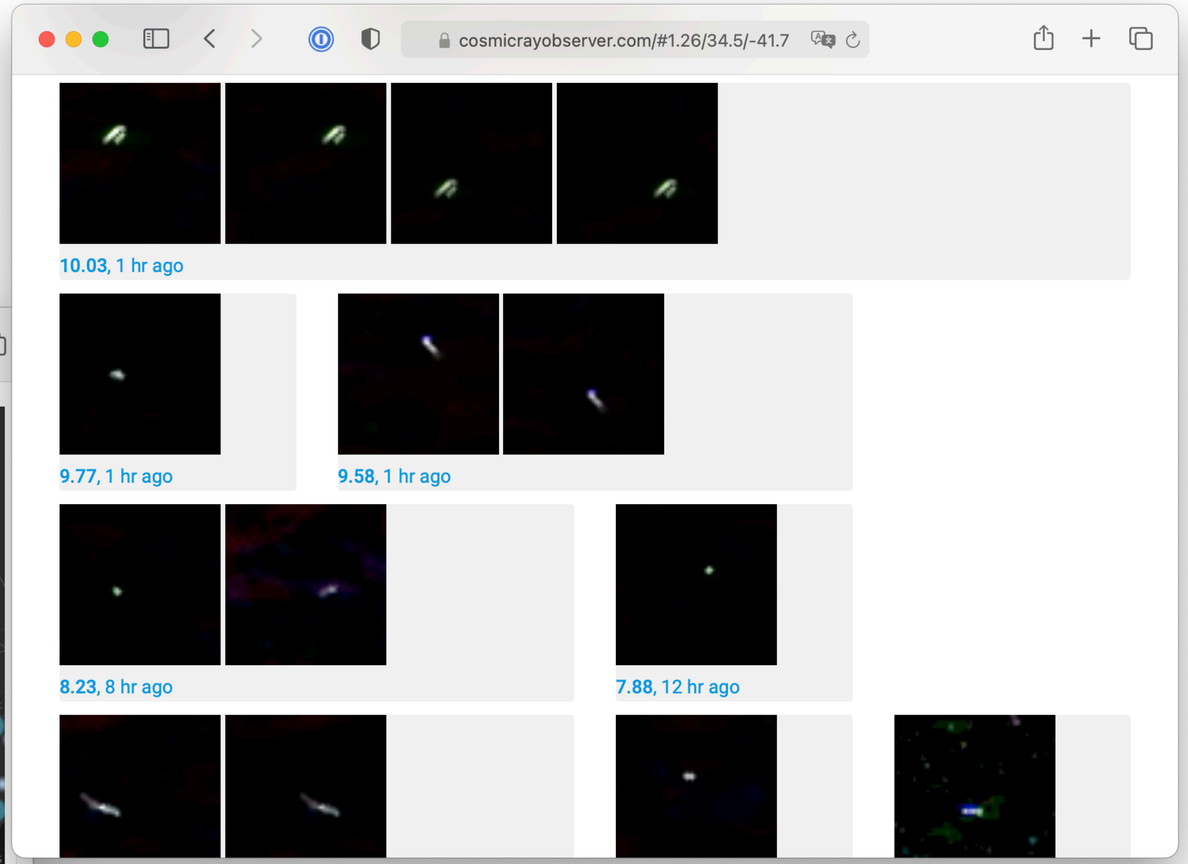
Abstract. The recent experimental proposals by Bose et al. and Marletto et al. (BMV) outline a way to test for the quantum nature of gravity by measuring gravitationally induced differential phase accumulation over the superposed paths of two $\sim 10^{-14}kg$ masses. These

- Built Starry Night in 1997 - 2021 (astronomy program)
- Software - other iOS, web apps, etc.
- Quantum Foundations conferences and papers

Cosmic Ray App

- Launched 2016 as iOS app.
- Server uploading of events started 2018
- 7 million events recorded.





cosmicrayobserver.com

Tom Andersen nSCIr.ca 2021

Cosmic Ray App

- iOS app is Objective-C - C - ‘Apple Metal’
- Uses standard Apple API, etc.
- Image analysis carried out in C or Metal (GPU).
- Image Captured as still from video mode at highest resolution
- 1 to ~5 frames per second.
- Each frame is analyzed by either C or Metal code (same math)
- Events are captured about once every 30s

Init Camera

- ask for video capture device
- register to get called back on frame ready

```
- (void) createSession {
    // create a capture session
    session = [[AVCaptureSession alloc] init];


    // setup the device and input
    AVCaptureDevice *videoCaptureDevice = [AVCaptureDevice defaultDeviceWithMediaType:AVMediaTypeVideo];
    NSError *error = nil;
    [GLCamera configureCameraForLowLight:videoCaptureDevice];
```

Warm up

- App captures ~50 frames on launch
- A heat map is created which gets the average firing level of every pixel, this data is subtracted off of captured images.
- Heat map is updated often (every frame in Metal)
- An assumption is made about thresholds for events, using if available the last threshold level used on the camera.

```
- (void)processNewCameraFrame:(CVImageBufferRef)cameraFrame {
    self.rayTime = [NSDate date];
    // http://stackoverflow.com/questions/4036737/how-to-draw-a-texture-as-a-2d-background-in-opengl-es-2-0
    CVPixelBufferLockBaseAddress(cameraFrame, 0);
    uint8_t* pixels = (uint8_t*)CVPixelBufferGetBaseAddress(cameraFrame);
```

Event capture

- self calibrating.
- app sets target to add one event per 30s
- for every frame calculate trigger blocks
- trigger blocks are 20x20 pixel sized blocks of the ~8 million pixel image.

```
// does not update the heatmap.
-(void)calculateTriggerBlocks:(uint8_t*)pixels;
{
#if DEBUG
    double start = [NSDate timeIntervalSinceReferenceDate];
#endif
    const long bufWidth = self.bufferWidth;
    const long bytesPerRow = self.bytesPerRow;
    float* heatMap = self.pixelHeatMap;
```

Event capture - trigger block math

- Skip 60 pixels along edges.
- For each block - calculate a score pixels scored

```
float blockScore = 0;
for (long countH = countTH*kTriggerZoneSize; countH < (countTH + 1)*kTriggerZoneSize; countH++)
{
    uint8_t* row = pixels + (countH * bytesPerRow);
    float* heatMapRow = heatMap + (countH*bufWidth);

    long startLoop = countTW*kTriggerZoneSize;
    long endLoop = (countTW + 1)*kTriggerZoneSize;

    for (long countW = startLoop; countW < endLoop; countW++)
    {
        uint8_t* pixel = row + (countW*4);
        long red = pixel[0];
        long green = pixel[1];
        long blue = pixel[2];
        long total = red + green + blue;
        blockScore += pixelScore(total, heatMapRow[countW]);
    }
}
triggerBlockRow[countTW] = blockScore;
```

Event capture - pixel score

- subtract heat map, only take pixels that are 4x over the heat map
- assume energy/score is diff*diff (does that make sense?)

```
float pixelScore(float total, float heat)
{
    float diff = total - heat;
    return (total > 4 && diff > 4*heat) ? diff*diff : 0.0f;
}
```

Event capture - analyze trigger blocks

- **Calculate largest highest scoring (and 2nd) blocks**

```
[self doTriggerBlockStats:theTriggerMap];
```

- for every block that exceeds threshold (0 - 30 blocks)
- calculate the brightest pixel in that block
- create an image buffer 3x3 blocks in size (so 60x60px)
- Use brightest pixel to set the scale.
- Keep RGB for fun.

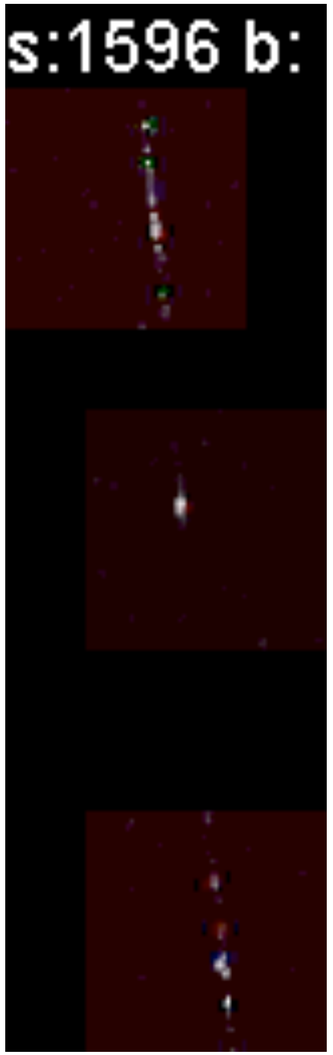
```
double brightness = 1.0/maxPixelComponent*sqrt(sqrt(maxPixelComponent/255.0));

uint8_t* pixel = row + (countW*4);

// mult by brightness, make unit 0  --> 1
double red = fmin(pixel[0]*brightness, 1.0);
double green = fmin(pixel[1]*brightness, 1.0);
double blue = fmin(pixel[2]*brightness, 1.0);

// use sqrt to get more detail from the lower lit pixels.
if (needLowLevelBoost)
{
    red = sqrt(red);
    green = sqrt(green);
    blue = sqrt(blue);
}
```

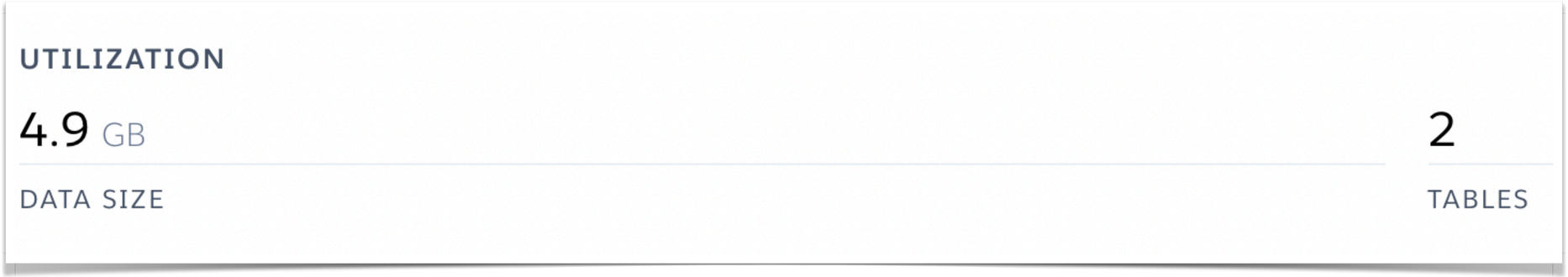
Image samples



Tom Andersen nSCIr.ca 2021

Image upload to server cosmicrayobserver.com

- Ignore events with too many lit blocks (over 30)
- Upload images and JSON data on each event
- One event can have more than one image
- Upload to PostgreSQL database, images to AWS S3
- Very simple data layout, two tables, images and events
- Web app written in Ruby Sinatra (very simple framework)
- Uses JS on web page to grab recent events
- Currently hosted on Heroku.com



Source code

- Source code available from Tom Andersen
- Not a public repository
- Free to use modify the code for CREDO purposes.
- Web Ruby/JS code also available.

Thanks