Luar Topics

- C – A Low-level Programming Language
- Make – A dependency-based build system
- YUYV – Representation of Color
- ZMP – Zero Moment Point control

C – A Low-level Programming Language

- Low-level
    - Compiles to machine code
    - No fancy semantics
    - No memory management
    - Strongly Typed, Simple Types
        - int, uint, float, double (no string, no bool)
    - Passes ”objects” by ”pointer”
- Function calls are based on ”signatures”
- Divided into .c and .h files

C Example

```c
// File: hello.c
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Hello World!\n");
}
```

Compile with:

```
gcc hello.c
```

Run with:

```
./a.out
```

C Example

```c
// File: hello.c
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Hello World!\n");
}
```

Compile with:

```
gcc hello.c -o hello
```

Run with:

```
./hello
```

C

- Strings are created out of an array of char
- char is really just an int
- There are no objects, so arrays don't know how long they are... you have to count the elements
- Arrays are passed to a function by their ”address” in memory, called a ”pointer”

C Example

```
// File: hello.c
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Hello World!\n");
}
```

C Example

```
// File: hello2.c
#include <stdio.h>  // for printf
#include <string.h> // for strlen, strcat
#include <stdlib.h> // for malloc, free

void say_hello(char * name) {
  char *result = malloc(strlen(name) + strlen("Hello ") + 2); // for '\n' and '\0'
  strcat(result, "Hello ");
  strcat(result, name);
  strcat(result, "\n");
  printf(result);
  free(result);
}

int main(int argc, char **argv) {
  int i;
  for (i = 0; i < argc; i++) {
    say_hello(argv[i]);
  }
}
```

C Details

- argv[i] is short for *(argv + i)
  - Pointer arithmetic
- Use malloc() to ”memory allocate” exact space that you need
- Use free() to release that memory back to the system
- Use strcat() to concatenate strings into a accumulator

C Example

```
// File: hello2.c
#include <stdio.h>  // for printf
#include <string.h> // for strlen, strcat
#include <stdlib.h> // for malloc, free

void say_hello(char * name) {
  char *result = malloc(strlen(name) + strlen("Hello ") + 2); // for '\n' and '\0'
  strcat(result, "Hello ");
  strcat(result, name);
  strcat(result, "\n");
  printf(result);
  free(result);
}

int main(int argc, char **argv) {
  int i;
  for (i = 0; i < argc; i++) {
    say_hello(argv[i]);
  }
}
```

C Example: hello3.c

```
// File: hello3.c
# include ”hellolib.h”

int main(int argc, char **argv) {
  int i;
  for (i = 0; i < argc; i++) {
    say_hello(argv[i]);
  }
}
```

C Example: hellolib.c

```
// File: hellolib.c
#include <stdio.h>  // for printf
#include <string.h> // for strlen, strcat
#include <stdlib.h> // for malloc, free

void say_hello(char * name) {
  char *result = malloc(strlen(name) + strlen("Hello ") + 2); // for '\n' and '\0'
  strcat(result, "Hello ");
  strcat(result, name);
  strcat(result, "\n");
  printf(result);
  free(result);
}
```

C Example: hellolib.h

```
// File: hellolib.h
void say_hello(char *);
```

C Example

- Source Files:
    - hello3.c – source with main()
    - hellolib.c – source defining library
    - hellolib.h – header file of library
- Compile in two steps:
    - Create a ”shared object” file:
        - `gcc -fPIC -shared hellolib.c -o hellolib.so`
    - Create the main, by compiling and linking:
        - `gcc hello3.c -o hello3 hellolib.so`

C Summary

- Use malloc() and free() to manage memory
- Variables and functions must be identified before use
- Users must attend to a level of detail not necessary in higher level languages
- Gain a level of control (and speed) not available in higher level languages
- Catastrophic failure if not handled properly
- .h files are for making the build system fast

Make – A Dependency-based Build System

- Dependencies
    - hello3 depends on hello3.c and hellolib.so
    - hellolib.so depends on hellolib.c
- Build
    - Compiling and linking

Make: Example

```
# File: Makefile

hello3: hello3.c hellolib.so
    gcc hello3.c -o hello3 hellolib.so

hellolib.so: hellolib.c
    gcc -fPIC -shared hellolib.c -o hellolib.so
```

Make: Example

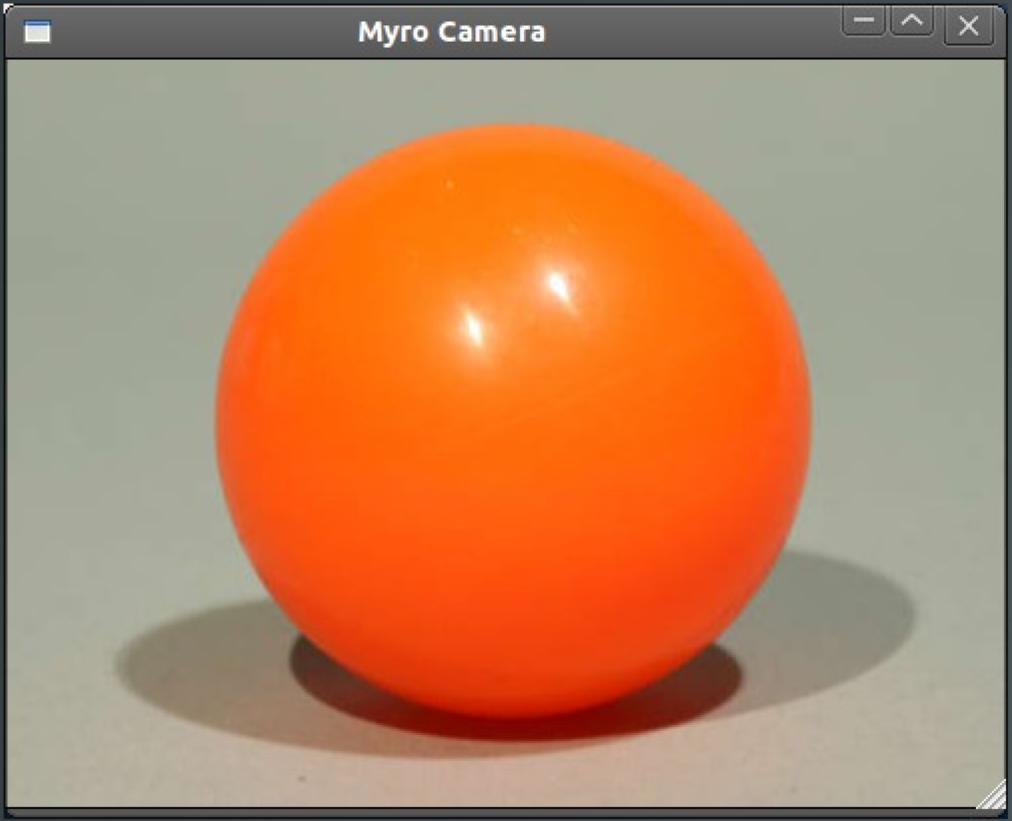
```
# File: Makefile

hello3: hello3.c hellolib.h hellolib.so
    gcc hello3.c -o hello3 hellolib.so

hellolib.so:
    gcc -fPIC -shared hellolib.c -o hellolib.so

clean:
        rm -rf *~ hellolib.so hello3
```

Make: Example

```
make
make clean
make -f Makefile.other
make install
```

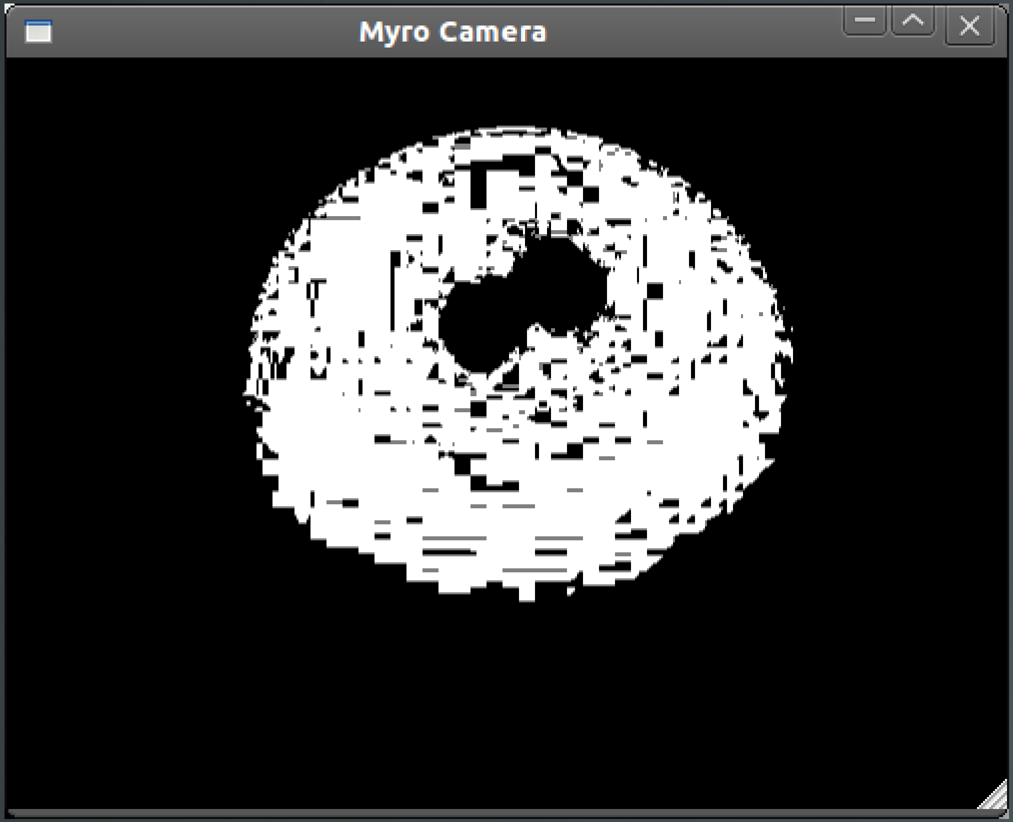
YUV – Representation of Colors

- RGB – not necessarily the best representation for fast image processing



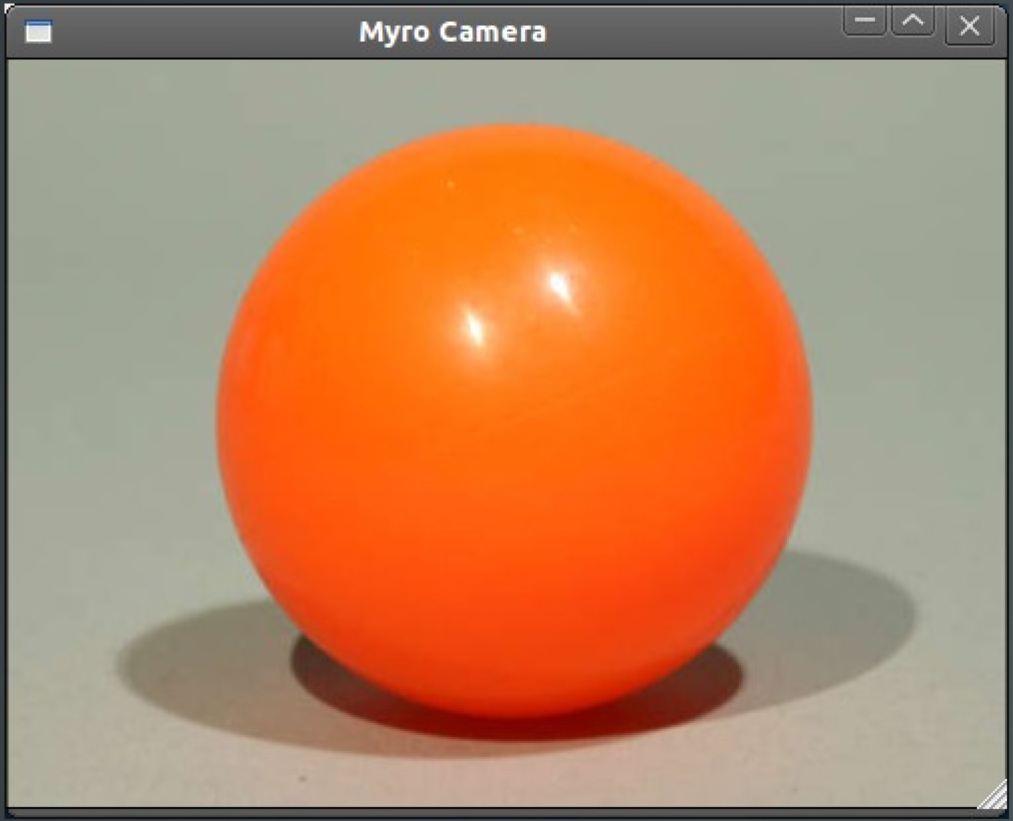
YUV – Representation of Colors

- Between (254, 40, 0) and (255, 140, 255)



YUV – Representation of Colors

- If we could separate out the *brightness* (or *luminance*) ...

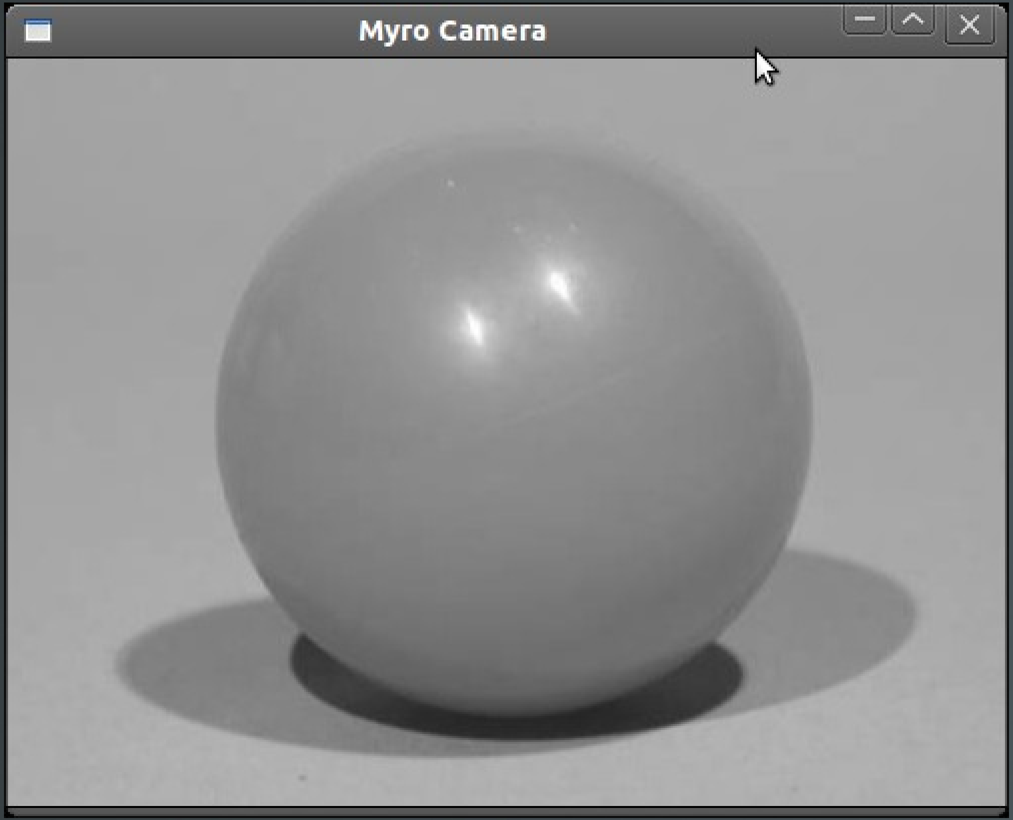


RGB to YUV

```
# Y'UV – one channel of brightness (Y'), two channels of color (U, V)

def rgb2yuv(picture):
    for pixel in getPixels(picture):
        R, G, B = getRGB(pixel)
        Y = min(max(int(0.299 * R + 0.587 * G + 0.114 * B), 0), 255)
        U = min(max(int(-0.169 * R - 0.332 * G + 0.500 * B + 128), 0), 255)
        V = min(max(int( 0.500 * R - 0.419 * G - 0.0813 * B + 128), 0), 255)
        setRGB(pixel, Y, Y, Y)
```

YUV – Representation of Colors

- If we could separate out the *brightness* (or *luminance*) we could see just the brightness

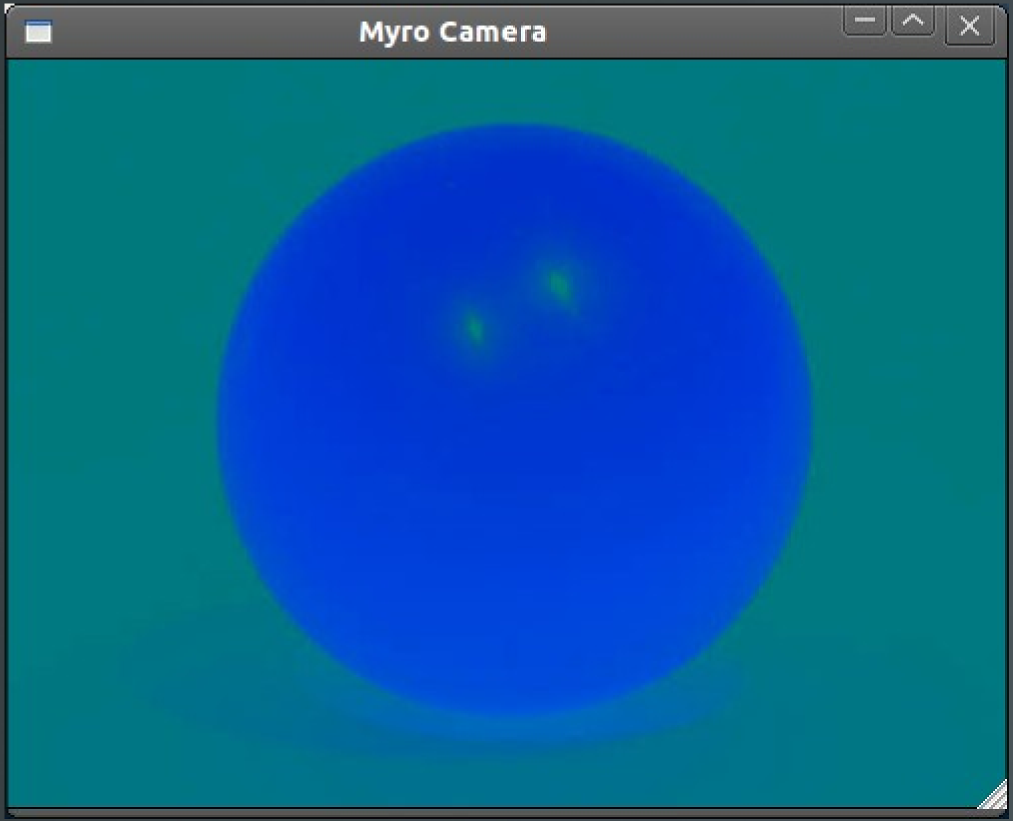


RGB to YUV

```
# Y'UV – one channel of brightness (Y'), two channels of color (U, V)

def rgb2yuv(picture):
    for pixel in getPixels(picture):
        R, G, B = getRGB(pixel)
        Y = min(max(int(0.299 * R + 0.587 * G + 0.114 * B), 0), 255)
        U = min(max(int(-0.169 * R - 0.332 * G + 0.500 * B + 128), 0), 255)
        V = min(max(int( 0.500 * R - 0.419 * G - 0.0813 * B + 128), 0), 255)
        setRGB(pixel, 0, U, V)
```

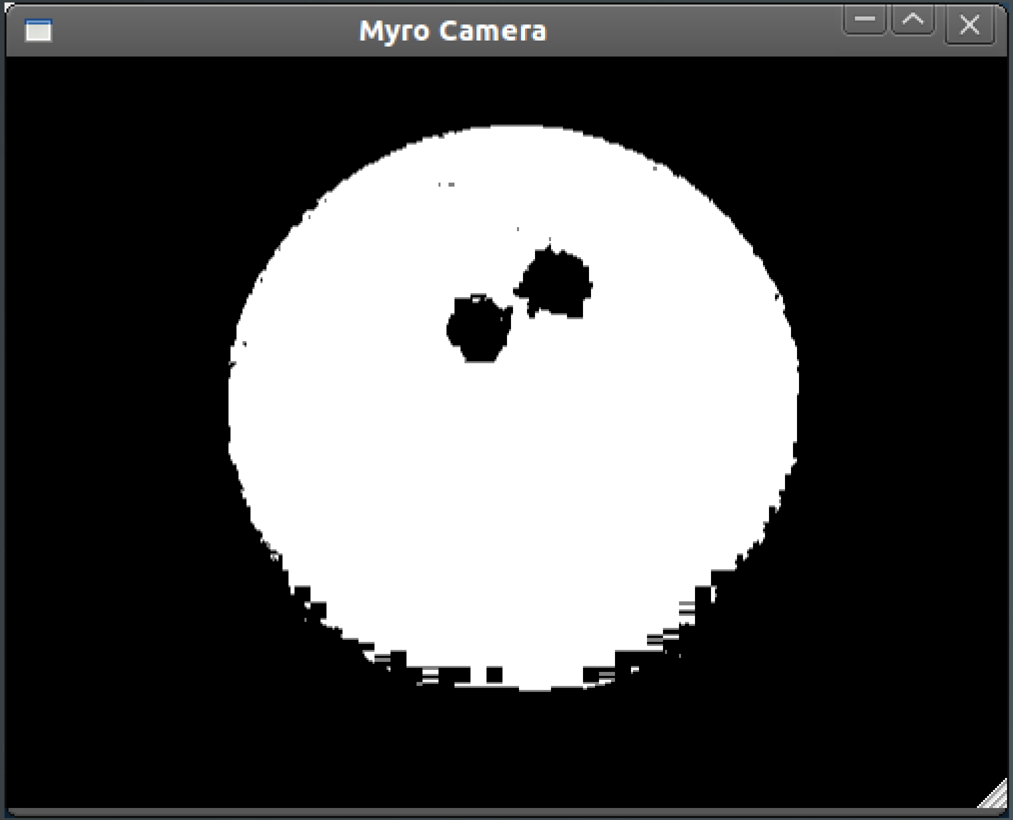
YUV – Representation of Colors

- Looks more like a two dimensional circle, than a three dimensional sphere



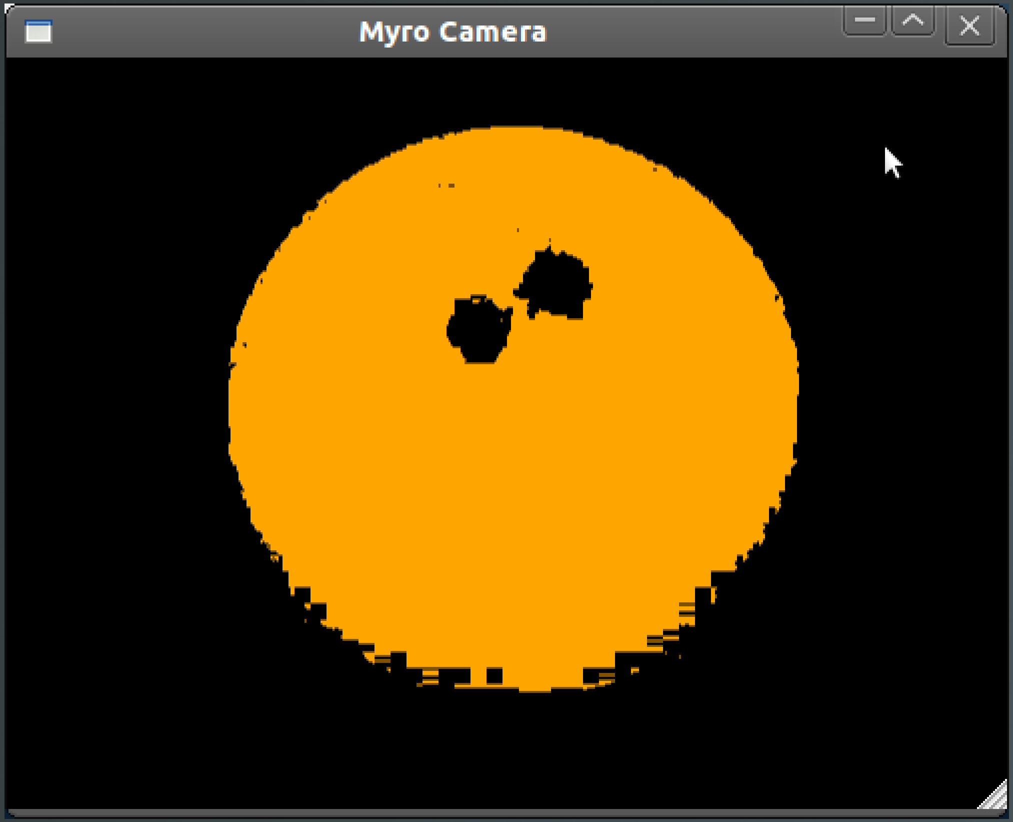
YUV – Representation of Colors

- Between (0, 45, 185) and (0, 75, 240)



YUV – Representation of Colors

- Between (0, 45, 185) and (0, 75, 240)
- Add *false color*



ZMP – Zero Moment Point Control

”The forces acting on a walker can be separated in two categories:

1) forces exerted by contact and
2) forces transmitted without contact (gravity and, by extension, inertia forces).

The Center of Pressure (CoP) is linked to the former, and the Zero Moment Point (ZMP) to the latter.”

Forces Acting on a Biped Robot: Center of Pressure—Zero Moment Point. 2004. Philippe Sardain and Guy Bessonnet