



Line 289 // Second Pass through Image, Threshold and initial equivalency

```
for (r=0;r<IMAGE_ROWS;r++) {
    for(c=0;c<IMAGE_COLUMNS;c++) {
        if ( (abs((unsigned int)HSV_Image[r][c].s-specs_s)<=specs_srad)
            && (abs((unsigned int)HSV_Image[r][c].v-specs_v)<=specs_vrad)
            && ( (abs((unsigned int)HSV_Image[r][c].h-specs_h)<=specs_hrad)
                || (abs((unsigned int)HSV_Image[r][c].h-specs_h2)<=specs_hrad) // catch the hue wraparound
            )
        ) {
            object_detected = 1; // Set a flag that at least one pixel found above threshold

            // ----- Connectivity calculations -----
            // Labels pixels 1 to MAX_NUM_EQUIVALENCIES depending on top and left neighbors
            // The labels represent object number...
            if (r == 0) top = 0; else top = Thres_Image[r-1][c]; // previous row, same column

            if (c == 0) left = 0; else left = Thres_Image[r][c-1]; // same row, previous column

            neighbor_type = 0;
            if (left != 0) neighbor_type += 1;
            if (top != 0) neighbor_type += 2;

            current_object = 0;
            switch (neighbor_type) {
                case 0: // Both neighbors zero, New object needed
                    if (num_unique_objects < (MAX_NUM_EQUIVALENCIES-1) ) {
                        num_unique_objects++;
                    } else {
                        too_many_objects++;
                    }
                    equivalency_objects[num_unique_objects] = num_unique_objects;
                    current_object = num_unique_objects;
                break;
                case 1: // Top is zero, left is not zero
                    current_object = left;
                break;
                case 2: // Left is zero, top is not zero
                    current_object = top;
                break;
                case 3: // Top and left are not zero... must note equivalency
                    if (top == left) current_object = left;
                    else {
                        if (Check_Equivalency(top,left) == 0) {
                            current_object = Set_Equivalency(top,left);
                        } else {
                            current_object = left;
                        }
                    }
                break;
                default: // Should NEVER enter here
                    current_object = 0; // Object 0 stores errors
                break;
            }

            Thres_Image[r][c] = current_object;
            object_stats[current_object].num_pixels_in_object +=1;
            object_stats[current_object].sum_r += r;
            object_stats[current_object].sum_c += c;
            // ----- Done with connectivity calculations (first pass) -----
        } else {
            Thres_Image[r][c] = 0;
        }
    } // end for loop on c
} // end for loop on r
```

```
// No reason for these to be global except that it is easier to debug/view these variables in CCS when global
```

```
int current;  
int equivalency = 0;  
int done = 0;
```

```
int Check_Equivalency(int A, int B) {      // Looks through the link array starting at A  
                                           // to see if A and B are equivalent  
    done = 0;  
    equivalency = 0;  
    current = equivalency_objects[A];  
    while (done == 0) {  
        if (current == A) {  
            done = 1;  
        }  
        else {  
            if (current == B) {  
                equivalency = 1;  
                done = 1;  
            }  
            else current = equivalency_objects[current];  
        }  
    }  
    return (equivalency);  
}
```

```
int Set_Equivalency (int A, int B) {  
    // Modifies link list so that A and B are equivalent  
    // and returns lower of A and B  
    // Operations:  
    //          1: temp set to value at A  
    //          2: A set to value at B  
    //          3: B set to temp  
    // NOTE: Does NOT work if equivalence between A and B already done!  
    int temp = equivalency_objects[A];  
  
    equivalency_objects[A] = equivalency_objects[B];  
    equivalency_objects[B] = temp;  
    if (A > B) return(A);  
    else return(B);  
}
```

```
int Fix_Equivalency(int num_equivalencies_used) { // Fixes equivalency to ordered values, returns num. objects..
```

```
    int i;
    int ordered_num = 1;
    int num_unique = 0;
    int done = 0;
    int current = 0;

    int num_equivs = num_equivalencies_used+1;

    // just in case invalid parameter sent
    if (num_equivs > MAX_NUM_EQUIVALENCIES) num_equivs = MAX_NUM_EQUIVALENCIES;
    if (num_equivs < 1) num_equivs = 1;

    // zero temp link array
    for (i=1; i < num_equivs; i++) {
        temp_equivalency_objects[i] = 0;
    }

    for (i=1; i < num_equivs; i++) {

        if ( (temp_equivalency_objects[i] == 0) && (equivalency_objects[i] != 0) ) {
            temp_equivalency_objects[i] = ordered_num;
            ordered_num++;
            done = 0;
            current = equivalency_objects[i];
            while (done == 0) {
                if (current == i) done = 1;
                else {
                    if (temp_equivalency_objects[current] == 0) {
                        temp_equivalency_objects[current] = temp_equivalency_objects[i];
                    }
                    current = equivalency_objects[current];
                }
            }
        }
    }

    num_unique = ordered_num - 1;

    // Copy equivalencies over
    for (i=1; i < num_equivs; i++) {
        equivalency_objects[i] = temp_equivalency_objects[i];
    }

    if ( num_unique > MAX_NUM_OBJECTS ) {
        num_unique = MAX_NUM_OBJECTS;
    }

    // Add up totals, since many different objects now refer to same final object
    for (i=1; i < num_equivs; i++) {
        if (equivalency_objects[i] <= num_unique) {
            final_object_stats[equivalency_objects[i]].num_pixels_in_object += object_stats[i].num_pixels_in_object;
            final_object_stats[equivalency_objects[i]].sum_r += object_stats[i].sum_r;
            final_object_stats[equivalency_objects[i]].sum_c += object_stats[i].sum_c;
        }
    }

    // Calculate the average pixel values
    for (i=1; i <= num_unique; i++) {
        if (final_object_stats[i].num_pixels_in_object != 0) {
            final_object_stats[i].center_r = final_object_stats[i].sum_r/final_object_stats[i].num_pixels_in_object;
            final_object_stats[i].center_c = final_object_stats[i].sum_c/final_object_stats[i].num_pixels_in_object;
        }
    }

    // Now have ordered_num-1 unique objects
    return(num_unique);
}
```