Change Part

custom structure `byd_interfaces_cam_od_list`

```c
typedef struct byd_interfaces_msg_CamObjType
{
  uint32_t id;
  float objConf;
  float x0PixPoint;
  float y0PixPoint;
  float x1PixPoint;
  float y1PixPoint;
  float objClassConfidence;
} byd_interfaces_msg_CamObjType;

typedef struct
{
  uint32_t frame_id;
  uint64_t timestamp;
  uint8_t status;
  uint32_t od_num;
  byd_interfaces_msg_CamObjType od_list[32];
} byd_interfaces_cam_od_list;
```

change part：

The red part is where I modified.

1. app_post_proc_module.c

```
vx_status app_init_post_proc(vx_context context, PostProcObj *postProcObj, char *objName)
{
 .......

  // od_data: Creates a reference to a User Data Object
  postProcObj->vx_od_data = vxCreateUserDataObject(context, "", sizeof(byd_interfaces_cam_od_list), NULL);
  APP_ASSERT_VALID_REF(postProcObj->vx_od_data);

  if(status == VX_SUCCESS)
  {
    //  associate a name to a referenced object, 用于log
    snprintf(ref_name, APP_MAX_FILE_PATH, "%s_vx_od_data", objName);
    status = vxSetReferenceName((vx_reference)postProcObj->vx_od_data, ref_name);
  }

  ..........
}

void app_create_graph_post_proc(vx_graph graph,
                    PostProcObj *postProcObj,
                    // vx_object_array input_image_arr,
                    byd_interfaces_cam_od_list* od_data,
                    vx_object_array out_args_arr,
                    vx_object_array output_tensors_arr)
{
  vx_user_data_object out_args;
  vx_tensor output_tensors[APP_MAX_TENSORS];
  vx_image output_images[APP_MAX_TENSORS];
  vx_int32 i;

  vx_status status = VX_SUCCESS;
  status = vxCopyUserDataObject(postProcObj->vx_od_data, 0, sizeof(byd_interfaces_cam_od_list),
          &postProcObj->od_data, VX_WRITE_ONLY, VX_MEMORY_TYPE_HOST);
  printf("status = %d\n", status);

  out_args = (vx_user_data_object)vxGetObjectArrayItem((vx_object_array)out_args_arr, 0);

  output_tensors[0] = (vx_tensor)vxGetObjectArrayItem((vx_object_array)output_tensors_arr, 0);
  output_images[0] = (vx_image)vxGetObjectArrayItem((vx_object_array)postProcObj->output_image_arr, 0);

  postProcObj->node = tivxPixelVizNode(graph,
                    postProcObj->viz_kernel,
                    postProcObj->viz_config,
                    out_args,
                    postProcObj->vx_od_data, // change !!!!!
                    postProcObj->num_output_tensors,
                    output_tensors,
                    output_images);
```

```
    APP_ASSERT_VALID_REF(postProcObj->node);

    vxSetNodeTarget(postProcObj->node, VX_TARGET_STRING, TIVX_TARGET_DSP2);
    vxSetReferenceName((vx_reference)postProcObj->node, "PostProcNode");

    vx_bool replicate[32];

    replicate[0] = vx_false_e;
    replicate[1] = vx_true_e;
    replicate[2] = vx_true_e;
    for(i = 0; i < (postProcObj->num_output_tensors * 2); i++)
    {
      replicate[3 + i] =  vx_true_e;
    }

    // get error
    vxReplicateNode(graph, postProcObj->node, replicate, (3 + postProcObj->num_output_tensors*2));

    vxReleaseUserDataObject(&out_args);

    for(i = 0; i < postProcObj->num_output_tensors; i++)
    {
      vxReleaseTensor(&output_tensors[i]);
      vxReleaseImage(&output_images[i]);
    }
}
```

2. kernels/img_proc/include/TI/tivx_img_proc_nodes.h

```
VX_API_ENTRY vx_node VX_API_CALL tivxPixelVizNode(vx_graph graph,
                    vx_kernel         kernel,
                    vx_user_data_object  configuration,
                    vx_user_data_object  tidl_out_args,
                   //  vx_image         in_img,
                    vx_user_data_object  od_data,
                    vx_int32          num_output_tensors,
                    vx_tensor         detection[],
                    vx_image          out_img[])
```

3. kernels/img_proc/host/tivx_img_proc_node_api.c

```
VX_API_ENTRY vx_node VX_API_CALL tivxPixelVizNode(vx_graph         graph,
                                                  vx_kernel        kernel,
                                                  vx_user_data_object  configuration,
                                                  vx_user_data_object  tidl_out_args,
                                                  //  vx_image         in_img,
                                                  vx_user_data_object od_data,
                                                  vx_int32         num_output_tensors,
                                                  vx_tensor        detection[],
                                                  vx_image         out_img[])
{
  vx_reference prms[TIVX_KERNEL_PIXEL_VISUALIZATION_MAX_PARAMS];
  vx_int32 i;

  vx_int32 num_params = TIVX_KERNEL_PIXEL_VISUALIZATION_BASE_PARAMS + (num_output_tensors * 2);

  prms[0] = (vx_reference)configuration;
  prms[1] = (vx_reference)tidl_out_args;
  // prms[2] = (vx_reference)in_img;
  prms[2] = (vx_reference)od_data;

  for(i = 0; i < num_output_tensors; i++){
    prms[TIVX_KERNEL_PIXEL_VISUALIZATION_OUTPUT_TENSOR_IDX + i] = (vx_reference)detection[i];
    prms[TIVX_KERNEL_PIXEL_VISUALIZATION_OUTPUT_TENSOR_IDX + num_output_tensors + i] = (vx_reference)out_img[i];
  }

  vx_node node = tivxCreateNodeByKernelRef(graph,
                                           kernel,
                                           prms,
                                           num_params);
  return(node);

}
```

4. kernels/img_proc/host/tivx_pixel_visualization_host.c

```
  if (status == VX_SUCCESS)
      {
        status = vxAddParameterToKernel(kernel,
                 index,
                 VX_INPUT,
                 // VX_TYPE_IMAGE,
                 VX_TYPE_USER_DATA_OBJECT,
                 VX_PARAMETER_STATE_REQUIRED);
        index++;
      }
```