Converting to ARToolKit4
Part 1: Converting simpleLite.c

Philip Lamb, ARToolworks, Inc.
philip.lamb@artoolworks.com
2006-05-29

ARToolKit 4 incorporates several significant changes from ARToolKit 2.7x. It is much more object oriented (OO). Variables which were previously held in global static storage (inside the toolkit libraries) are now generally stored in structures, which the calling program must create. store, supply and free, when appropriate.

For a program based on simpleLite from ARToolKit 2.x, there are only a few modifications required. simpleLite is, as its name suggests, simple. It acquires images from only one camera, and loads a pattern file for only one marker. The simpleLite example does not demonstrate the advantages of ARToolKit 4's OO model. However, other more complex examples are based on the same code base as simpleLite, such as twoView (which acquires images from more than one camera) and simpleVRML (which loads pattern files for more than one marker). So we will begin with the changes between simpleLite from ARToolKit 2.72 and ARToolKit 4.x.

Follow the code of simpleLite.c as you read the changes below. The right-hand column indicates the location in the old simpleLite.c where the changes occur. The left hand column contains the new code that replaces the old code.

Global scope

We have three new global variables, and we no longer need the pattern center to be specified:

New	Old

// Marker detection.
static ARHandle *gARHandle = NULL;
static ARPattHandle *gARPattHandle = NULL;
static int gARTThreshhold = 100;
	// Marker detection.
	static int gARTThreshhold = 100;

// Transformation matrix retrieval.
static AR3DHandle *gAR3DHandle = NULL;
static double gPatt_width = 80.0; // Per-marker, but we are using only 1 marker.
static double gPatt_trans[3][4]; // Per-marker, but we are using only 1 marker.
	// Transformation matrix retrieval.
	static double gPatt_width = 80.0; // Per-marker, but we are using only 1 marker.
	static double gPatt_centre[2] = {0.0, 0.0}; // Per-marker, but we are using only 1 marker.
	static double gPatt_trans[3][4]; // Per-marker, but we are using only 1 marker.

setupCamera()

Global AR toolkit state is now held by the calling program, and so the setup code changes a little. We are now required us to pass in an ARHandle and an AR3DHandle to hold the global state, and the arInitCparam() call is replaced by a sequence of calls. The default threshold value is also set here, and must be passed in. Since ARToolKit 4 now supports runtime specification of the pixel format, we query the video library as to what format pixels from the camera are in, and let the marker detection library know.

New	Old
static int setupCamera(const char *cparam_name, char *vconf, int threshhold, ARParam *cparam, ARHandle **arhandle, AR3DHandle **ar3dhandle)	static int setupCamera(const char *cparam_name, char *vconf, ARParam *cparam)
int xsize, ysize; int pixFormat;	int xsize, ysize;

// Find the size of the window.
if (arVideoGetSize(&xsize, &ysize) < 0) return (FALSE);
fprintf(stdout, "Camera image size (x,y) = (%d,%d)\n", xsize, ysize);
	// Find the size of the window.
	if (arVideoInqSize(&xsize, &ysize) < 0) return (FALSE);
	fprintf(stdout, "Camera image size (x,y) = (%d,%d)\n", xsize, ysize);

// Get the format in which the camera is returning pixels.
pixFormat = arVideoGetPixelFormat();
if (pixFormat < 0) {
fprintf(stderr, "setupCamera(): Camera is using unsupported pixel format.\n");
return (FALSE);
}
 	
// Load the camera parameters, resize for the window and init.
	// Load the camera parameters, resize for the window and init.

if ((*arhandle = arCreateHandle(cparam)) == NULL) {
fprintf(stderr, "setupCamera(): Error: arCreateHandle.\n");
return (FALSE);
}
if (arSetPixelFormat(*arhandle, pixFormat) < 0) {
fprintf(stderr, "setupCamera(): Error: arSetPixelFormat.\n");
return (FALSE);
}
if (arSetDebugMode(*arhandle, AR_DEBUG_ENABLE) < 0) {
fprintf(stderr, "setupCamera(): Error: arSetDebugMode.\n");
return (FALSE);
}
	arInitCparam(cparam);

if (arSetLabelingThresh(*arhandle, threshhold) < 0) {
fprintf(stderr, "setupCamera(): Error: arSetDebugMode.\n");
return (FALSE);
}
if ((*ar3dhandle = ar3DCreateHandle(cparam)) == NULL) {
fprintf(stderr, "setupCamera(): Error: ar3DCreateHandle.\n");
return (FALSE);
}	

setupMarker()

Another piece of global ARToolKit state, the array of loaded markers, is also now held by the calling program, so setupMarker now requires an ARHandle and an ARPattHandle to be passed in. Once the one marker used by simpleLite is loaded, the pattHandle is attached.

New	Old

static int setupMarker(const char *patt_name, int *patt_id, ARHandle *arhandle, ARPattHandle **pattHandle)
{
if ((*pattHandle = arPattCreateHandle()) == NULL) {
fprintf(stderr, "setupCamera(): Error: arPattCreateHandle.\n");
return (FALSE);
}
if ((*patt_id = arPattLoad(*pattHandle, patt_name)) < 0) {
fprintf(stderr, "setupMarker(): pattern load error !!\n");
return (FALSE);
}
arPattAttach(arhandle, *pattHandle);
	static int setupMarker(const char *patt_name, int *patt_id)
	{
	if((*patt_id = arLoadPatt(patt_name)) < 0) {
	fprintf(stderr, "setupMarker(): pattern load error !!\n");
	return (FALSE);
	}

debugReportMode()

Some mode variables are no longer available to query. (Query functions may be available in a future ARToolKit4 release.)

New	Old

	if(arFittingMode == AR_FITTING_TO_INPUT) {
	fprintf(stderr, "FittingMode (Z): INPUT IMAGE\n");
	} else {
	fprintf(stderr, "FittingMode (Z): COMPENSATED IMAGE\n");
	}

	if(arImageProcMode == AR_IMAGE_PROC_IN_FULL) {
	fprintf(stderr, "ProcMode (X) : FULL IMAGE\n");
	} else {
	fprintf(stderr, "ProcMode (X) : HALF IMAGE\n");
	}

	if(arTemplateMatchingMode == AR_TEMPLATE_MATCHING_COLOR) {
	fprintf(stderr, "TemplateMatchingMode (M) : Color Template\n");
	} else {
	fprintf(stderr, "TemplateMatchingMode (M) : BW Template\n");
	}
	
	if(arMatchingPCAMode == AR_MATCHING_WITHOUT_PCA) {
	fprintf(stderr, "MatchingPCAMode (P) : Without PCA\n");
	} else {
	fprintf(stderr, "MatchingPCAMode (P) : With PCA\n");
	}

Quit()

Extra cleanup work is required:

New	Old

arglCleanup(gArglSettings);
arPattDetach(gARHandle);
arPattDeleteHandle(gARPattHandle);
arVideoCapStop();
ar3DDeleteHandle(gAR3DHandle);
arDeleteHandle(gARHandle);
arVideoClose();
	arglCleanup(gArglSettings);
	arVideoCapStop();
	arVideoClose();

Idle()

Detected markers are now stored in the ARHandle structure, leading to a number of minor changes in the marker detection loop. Additionally, different types of marker detection are available, so arGetTransMat has been renamed to clarify which type it uses.

New	Old

ARUint8 *image;
double err;
 	
int j, k;
	ARUint8 *image;
	
	ARMarkerInfo *marker_info; // Pointer to array holding the details of detected markers.
	int marker_num; // Count of number of markers detected.
	int j, k;

if (arDetectMarker(gARHandle, gARTImage) < 0) {	if (arDetectMarker(gARTImage, gARTThreshhold, &marker_info, &marker_num) < 0) {

for (j = 0; j < gARHandle->marker_num; j++) {
if (gARHandle->markerInfo[j].id == gPatt_id) {
if (k == -1) k = j; // First marker detected.
else if (gARHandle->markerInfo[j].cf > gARHandle->markerInfo[k].cf) k = j; // Higher confidence marker detected.
}
}
	for (j = 0; j < marker_num; j++) {
	if (marker_info[j].id == gPatt_id) {
	if (k == -1) k = j; // First marker detected.
	else if(marker_info[j].cf > marker_info[k].cf) k = j; // Higher confidence marker detected.
	}
	}

err = arGetTransMatSquare(gAR3DHandle, &(gARHandle->markerInfo[k]), gPatt_width, gPatt_trans);	arGetTransMat(&(marker_info[k]), gPatt_centre, gPatt_width, gPatt_trans);

Display()

arVideoCapNext() is no longer required.

New	Old
	arVideoCapNext();

main()

We must pass the modified parameters to setupCamera() and setupMarker():

New	Old
if (!setupCamera(cparam_name, vconf, gARTThreshhold, &gARTCparam, &gARHandle, &gAR3DHandle)) {	if (!setupCamera(cparam_name, vconf, &gARTCparam)) {
if (!setupMarker(patt_name, &gPatt_id, gARHandle, &gARPattHandle)) {	if (!setupMarker(patt_name, &gPatt_id)) {

That's all there is to it! The next instalment in this guide will cover converting the twoView and simpleVRML examples.